UIUCDCS-R-74-621

EMULATION OF THE PDP-11 INSTRUCTION SET ON THE LOCKHEED SUE PROCESSOR

by

James Fredrick Hart

April 1974

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-74-621

EMULATION OF THE PDP-11 INSTRUCTION SET ON THE LOCKHEED SUE PROCESSOR

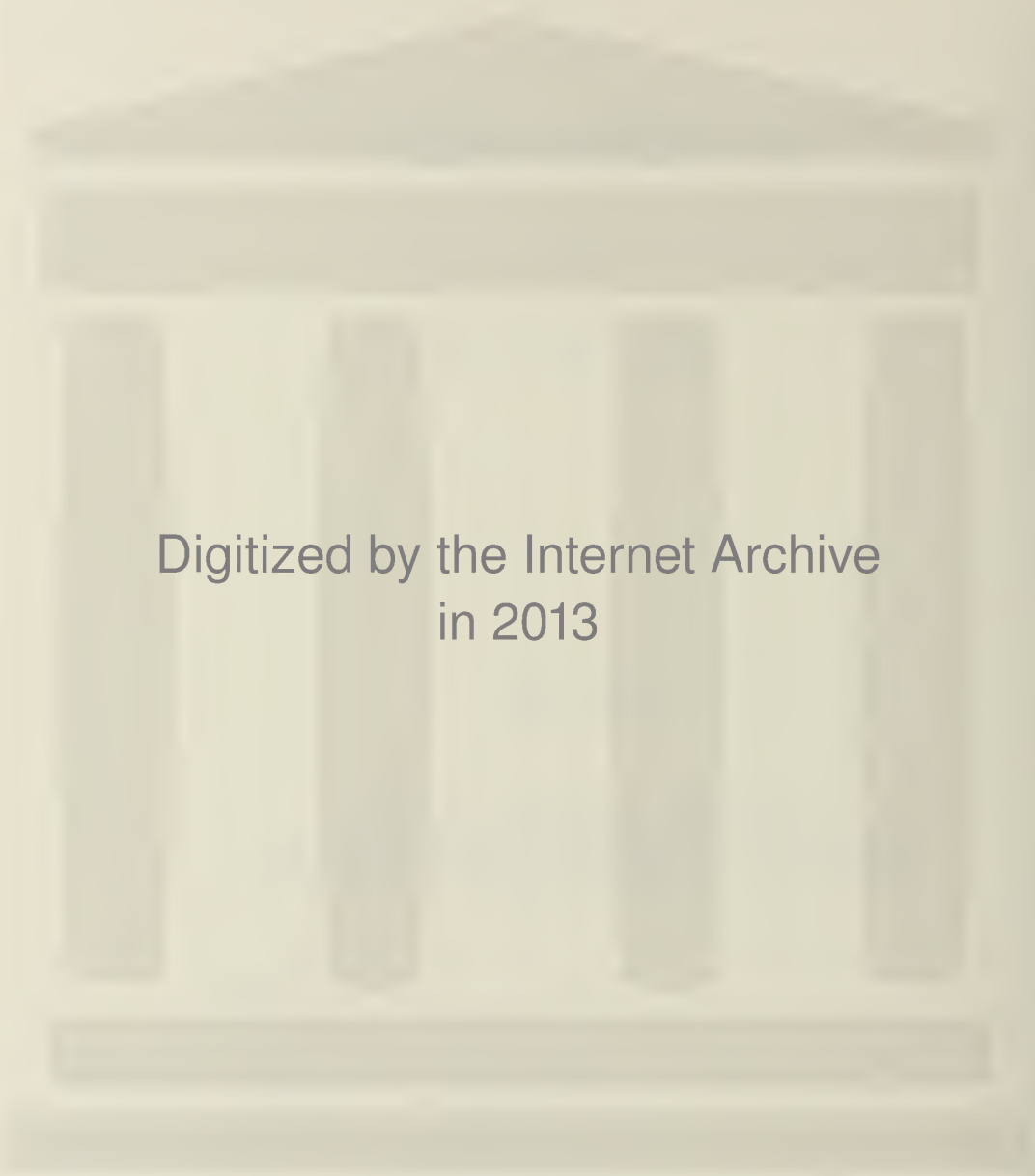
by

James Fredrick Hart

April 1974

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois

This work was supported in part by the National Science Foundation under Grant No. US NSF GJ-36265 and was submitted in partial fulfillment for the Master of Science degree in Computer Science, 1974



ACKNOWLEDGMENT

My sincere thanks to Professor James Vander Mey, my thesis advisor, who provided guidance and many helpful suggestions throughout the development of this thesis and Professor Donald Gillies who provided employment as a research assistant during the time I was a graduate student.

Thanks also go to my fellow students, especially Bill Tao who wrote a SUE microprogram simulator and Chuck Collins who provided a microassembler, and Mrs. June Wingler who performed an excellent job of typing this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

During the past several years the popularity of the low priced minicomputer has grown extensively. Of all the minicomputers available, the PDP-11 with its extensive instruction set and elegant stack and interrupt programming capabilities has become one of the most popular of the minicomputers. With the large amount of software presently available for this machine it becomes a natural choice as a base machine for a research project.

This thesis is the result of a project aimed at the investigation of networks consisting of minicomputers performing specialized functions. Some of specialization is to be performed in microcode, thus one restriction on any minicomputer which would be an active part of the network is that it be microprogrammable. Since it was necessary to perform functions such as simulation for debugging microprograms, assembling of microprograms, and simulation of algorithms to be used by various nodes on the network it was also necessary to have general purpose machines available. The machine chosen for this purpose was a PDP-11 because of the programming expertise of many members of the research group and because of the large amount of software already written for it by the group. Ideally the PDP-11's would be microprogrammable so that later they could be microprogrammed into specialized nodes for the network.

While all of the later PDP-11's are microprogrammed, the microprogram structure is explicitly tied to the PDP-11's instruction set and would

not allow implementation of the specialized functions which would be performed by the nodes of the network. Thus it was decided to choose a machine whose microprogramming capabilities were flexible enough to allow a PDP-11 instruction set emulator to be written for it as well as other specialized firmware.

After considering several machines the Lockheed SUE was chosen. The SUE had several advantages over the other machines. The SUE's microprogramming capabilities are quite flexible and only loosely tied to the SUE's instruction set. It also has a single bus architecture similar to that of the PDP-11 with the added advantage that up to four processors may be placed on the same bus.

This thesis discusses the implementation of the PDP-11 instruction set emulator on the SUE. It consists of four chapters and three appendices. Chapter I describes the SUE microinstruction set and some minor changes made to it to allow implementation of the emulator. Chapter II gives a description of the PDP-11 instruction set implemented including several instructions not available on the PDP-11. Chapter III discusses the layout of the emulator itself, and Chapter IV summarizes possible hardware modifications which would significantly increase the speed of the emulator and decrease the amount of micro-code required.

CHAPTER I

THE SUE PROCESSOR

The Lockheed SUE minicomputer is described briefly in this chapter. The description is based on the The SUE Computer Handbook [1] and several internal documents provided by Lockheed Electronics [2,3,4]. When correctness of the information provided in these documents was in question the SUE 1110 Processor Logic Drawings [5] were used to verify or correct the information. A more detailed description of the SUE microinstruction format is given in Appendix B.

A. Organization

The Lockheed SUE minicomputer is a sixteen bit parallel processor. The control of the CPU is performed by a sequence of thirty-six bit microinstructions. Microinstruction cycle time is 130 nanoseconds for logical operations and 160 nanoseconds for arithmetic operations. The SUE has a single bus architecture with all memory and I/O device registers being accessed via an asynchronous bus designated the Infibus. The Infibus consists of eighteen address lines, sixteen data lines, and various control and power lines.

Logically the SUE can be divided into four sections: the arithmetic and logic unit, the register file, the Infibus interface, and the micro-program control. A data flow diagram of the CPU is given in Figure 1-1.

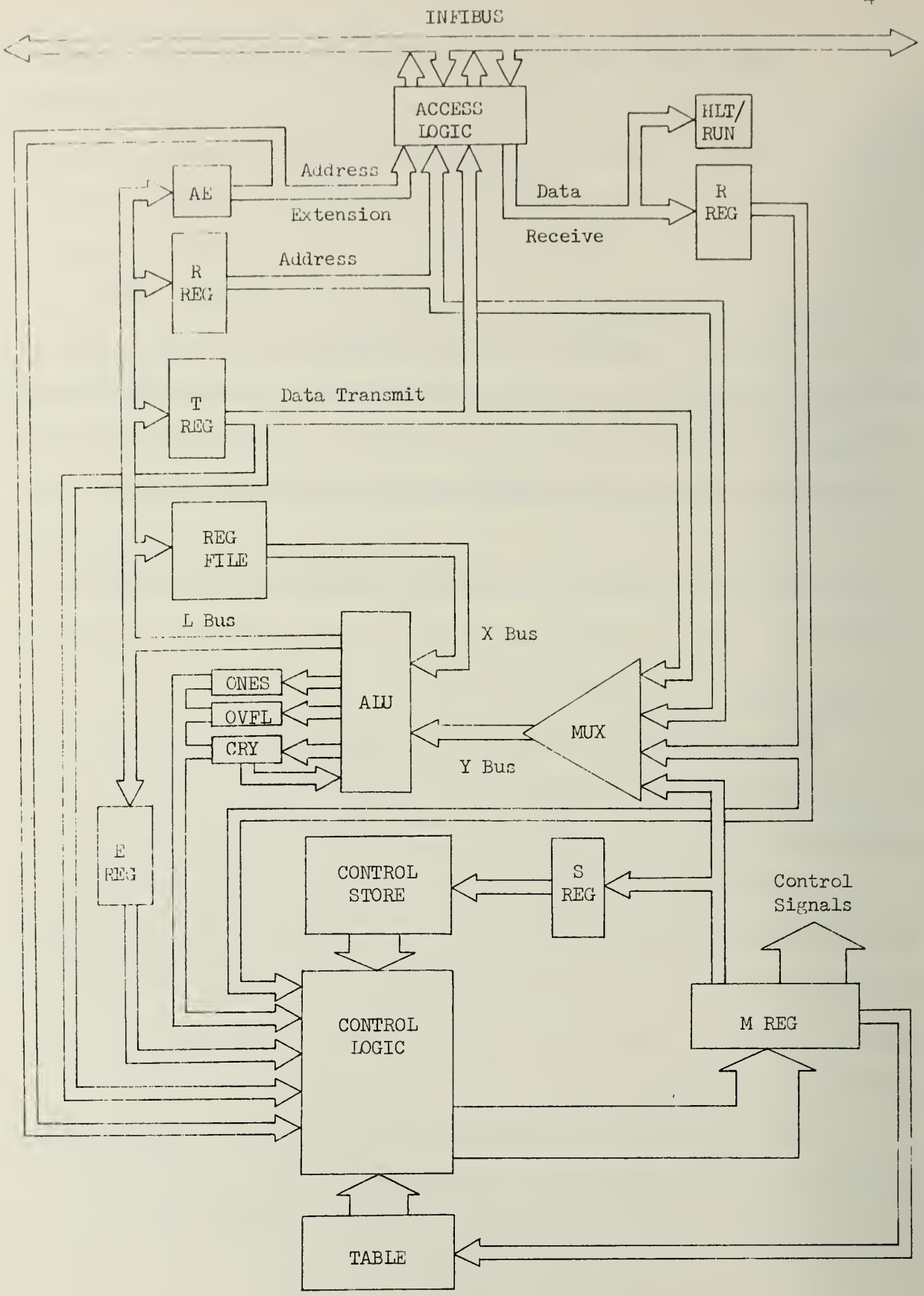


Figure 1.1 SUE Data Flow Diagram

Arithmetic and Logic Unit

The arithmetic and logic unit processes sixteen bits of data in parallel. It has two sixteen bit input data paths designated the XBUS and the YBUS and one sixteen bit output data path designated the LBUS. The XBUS data may come from one of twelve general purpose registers in the register file. The YBUS data is selected using a multiplexor. It may be the contents of the A, R, or T register, or a literal supplied from the literal field of the control word.

The ALU can perform one of thirty two functions, sixteen of the functions being logical with the other sixteen being arithmetic. The output may be used to set the carry and overflow flip flops if specified by the two bit C field in the microinstruction.

The output of the ALU may be transferred to the same file register specified as the XBUS input and also to either the E, T or A registers. A one bit register called the ones flip flop may also be set when the output of the ALU is all ones.

The Register File

The register file consists of a set of twelve sixteen bit registers which may be used as input to the ALU XBUS and may be loaded from the ALU LBUS. These registers may be used as general purpose temporary storage with the exception of register eight. This register is special in that it may only be used for a status register. Whenever it is loaded the top four bits are transferred to the bus controller and are used to mask interrupts from I/O devices.

Infibus Interface

The Infibus interface contains three sixteen bit registers as well as all necessary control logic and drivers to interface to the Infibus. The registers are the A, R and T registers. The A register is the address register. It contains the address of either a memory location or a device register which is to be accessed via the Infibus. The R or receive register receives all data being read via the Infibus. The R register may not be loaded by the microprogram, but only from the Infibus. The T register is the transmit register. It is used as its name implies to transmit data onto the Infibus. The T register also has another special function. It is a shift register which under microprogram control may perform one of eight different types of shifts.

Also included within the Infibus interface is a two bit address extension register. This register is an extension to the A register, and contains address bits sixteen and seventeen. It can be loaded from the two low order bits of the LBUS using a special command microstep.

Initiation of output to and input from Infibus is performed using special commands and completed by doing a wait for bus access completion microstep.

The Infibus interface also contains special logic to allow other master modules on the Infibus to access the register file while the CPU is halted. It also allows other master modules to access the CPU's halt and run flip flops so that they may halt and restart the CPU.

Microprogram Control

The microprogram control section of the SUE consists of the control store, a table containing rows of sixteen eight bit words used as branch

addresses or literal data, three registers and associated logic. The registers are the sixteen bit E register, the twelve bit S register and the thirty-six bit M register.

The control store consists of up to 1024 thirty-six bit words divided into pages of 256 words. The 1024 word limit is a physical one and by moving the control memory to a separate board it could be expanded to 4096 words.

The table is normally 256 words of eight bits each. However, a slight hardware modification will allow larger table sizes. The table is organized in rows of sixteen words. The entry selected from within a row is determined by the contents of one of four bit fields selected from the E register. The table is used extensively for instruction decoding.

The E register is used for encoding the next microinstruction. One of two four bit fields may be ANDed to the X field of the next microinstruction, and one of four four bit fields may be used to select a table row entry. The least significant four bits of the E register also have a special purpose. These four bits may be used as a loop counter which is incremented each time the T register is shifted by a special command.

The S or sequence register is the microprogram instruction counter. It contains the address of the next microinstruction to be executed. It may be modified only by performing branch or jump microinstructions. There are no provisions to save and restore the S register for the purpose of returning from microprogram subroutines.

The M register contains the encoded microinstruction presently being executed.

B. Microcode Description

A SUE microinstruction is divided into thirteen fields. The format of the microinstructions is given in Figure 1.2. A brief description of each field is given in this chapter, and a detailed description is given in Appendix B.

S Field

The S field is the microinstruction sequence control. It may specify that the microinstruction is to be sequential, sequential with a special command, a wait for an Infibus access to complete, an unconditional branch or a conditional jump.

T Field

The T field specifies the type of ALU function which is to be performed, that is, whether the function is logical or arithmetic.

A Field

The A field designates one of sixteen logical operations or one of sixteen arithmetic functions to be performed depending on the value of the T field.

C Field

The C field controls setting of the carry and overflow flip flops and the enabling of the carry bit to the carry input to the ALU. The actual meaning of the C field varies depending on whether the ALU operation is an arithmetic or logical one as determined by the T field. If it is logical the C field allows the carry and overflow flip flops to be cleared and the carry set. If it is arithmetic the C field designates whether the

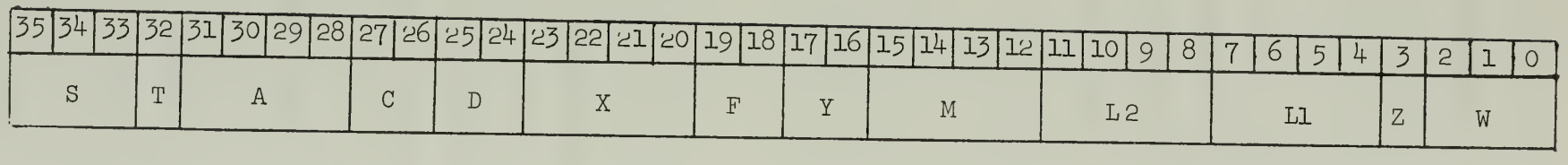


Figure 1.2 SUE Microcode Format

carry and overflow flip flops are to be set according to the results of the operation and also if the carry flip flop is to be enabled as input to the ALU.

D Field

The D field is used to determine whether or not the next microinstruction is to be executed. This is necessary because the next microinstruction is being fetched while the previous one is being executed. Thus on branch and jump microinstructions the actual branch or jump does not occur on the next microinstruction but on the one following it. The D field gives the microprogrammer the choice of whether or not to execute the next microinstruction. If it is to be skipped the M register is simply cleared and the only effect of that microinstruction is that the ONES flip flop is cleared. A full 130 nanoseconds is required whether the microinstruction is executed or not.

The meaning of the D field differs slightly depending on whether the S field specified a conditional jump or one of the other types of microinstructions. If it was a conditional jump the D field may specify whether to always execute the next microinstruction, never execute it, execute it only if the jump was performed, or execute it only if the jump wasn't performed. If the S field did not specify a conditional jump the D field allows the microprogrammer to always execute the next microinstruction, to never execute it, to execute it only if the output of the ALU was all ones, or to execute it only if the output of the ALU wasn't all ones. These choices allow tests to be made using ALU operations and to skip or execute the next microinstruction depending on the results of the test.

X Field

The X field selects which of the twelve file registers is to be gated to the XBUS for input to the ALU. This field may be modified according to the F field of the previous microinstruction.

F Field

The F field specifies modification of the next microinstruction X field. It can be modified by ANDing either a four bit field of the E register to it or by ANDing Infibus address bits one through four to it. The later case is used during slave address recognition.

Y Field

The Y field is used to multiplex data from one of three registers or from the literal fields to the YBUS. The registers are the R, A and T registers.

M Field

The use of the M field is determined by several other fields. These are the S field, the Y field, and if an extended table is used, the Z field.

If the S field specifies a branch microinstruction the most significant bits of the branch address are supplied by the M field. If the S field specifies a conditional jump the M field is used to specify the condition on which the jump is to be performed. If a shift is being performed the least significant three bits of the M field are used to specify the type of shift to be performed.

If the Y field specifies a literal the M field determines how the eight bit literal is to be mapped onto the sixteen bit YBUS.

If table data is to be obtained and the processor has been modified for an extended table the most significant bits of the table address are specified by the M field.

L2 Field

The L2 field is the left half of the eight bit literal field. When used as a literal it may be mapped into bits twelve through fifteen and bits four through seven of the YBUS depending on the value of the M field. When the S field specifies a special command the L2 field determines which special command is to be performed and if the Z and L1 field specify table data the L2 field specifies the four least significant bits of the row within the table.

L1 Field

The L1 field is the right half of the eight bit literal field. When used as a literal it may be mapped to bits eight through eleven or zero through three of the YBUS depending on the value of the M field. The L1 field may be modified by ANDing the carry, overflow, and ones flip flops, the extended address bits, and bits fifteen and zero of the T register. This can be done using special commands.

If the Z field specifies a special literal then the two least significant bits determine the type of special literal. For some special literals the most significant two bits specify which four bit field of the E register is to be used in determining the special literal.

Z Field

Determines whether or not the L fields specify a special literal.

W Field

The W field is used to select which registers the output of the ALU is to be written into. It may specify that the output is to be written into the file register as well as either the A, E or T registers. It also specifies when the ones flip flop is to be set.

CHAPTER II

INSTRUCTION SET DESCRIPTION

The PDP-11 contains one of the most comprehensive instruction sets ever available on a minicomputer. It consists of over eighty different instructions, many of which allow the user up to eight different addressing modes. This chapter describes the instruction set as emulated on the SUE. A more detailed description can be found in Appendix C and in the PDP-11 handbooks [6,7,8].

A. Registers

The PDP-11 contains eight general registers. Registers six and seven, however, are restricted in their use. Register seven is the program counter and contains the address of the next instruction to be executed. Register six is the system stack pointer. The stack is first in last out and is manipulated using various addressing modes, by interrupts, and by subroutine calls.

B. Processor Status Word

Also accessible to the program is the processor status word which may be addressed as a memory location by a program running in kernal mode. The status word as used by the emulator is shown in Figure 2.1. Its format is nearly identical to that used by the PDP-11/40 with the exception that bit 11 which specifies the general register set on the PDP-11/40 is

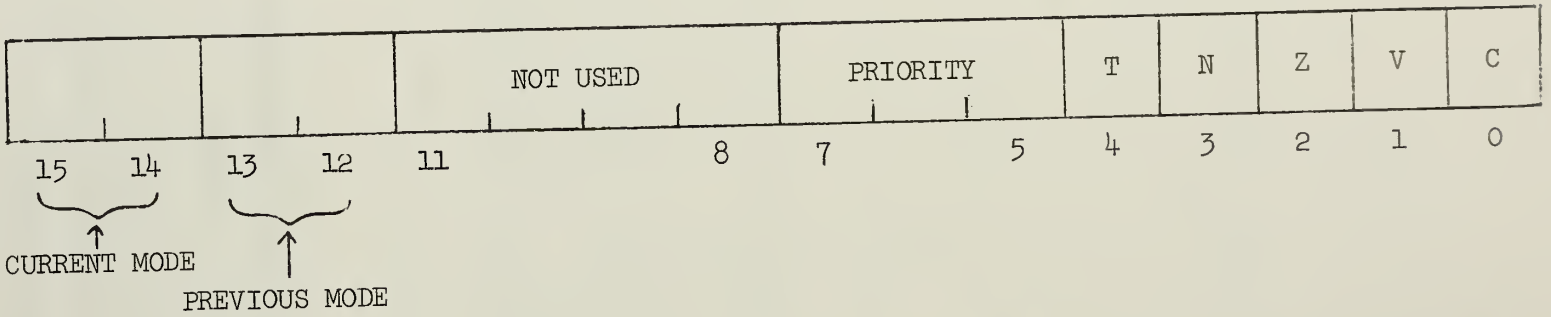


Figure 2.1 Program Status Word

not used. A description of each of the fields as used by the emulator is given below.

Current Mode - Two bit field specifying the current value of address bits 16 and 17.

Previous Mode - Two bit field specifying the previous value of address bits 16 and 17.

Priority - Three bit field containing the processor priority level. A device may not interrupt the processor unless its priority is greater than that given in the priority field of the program status word.

Trace - One bit field which when set will cause a software interrupt through location 14_8 of kernal mode memory after the completion of the current instruction.

Condition Codes - Four bit field containing the condition codes. They may be set automatically by instruction execution or set directly by accessing the processor status or by executing a condition code instruction.

C. Interrupts

Interrupts may be initiated by external hardware devices, by software through the execution of one of several special interrupt instructions, or internally by detection of the trace bit being set, an access to a nonexistent memory location, or by an attempt to execute an illegal instruction.

When an interrupt occurs the processor status is pushed onto the system stack followed by the current program counter. The program counter and status word are then loaded from two consecutive memory locations in

kernal address space as specified by the interrupting device or internally by the processor. When leaving the interrupt service routine a return from interrupt instruction may be executed which will restore the old program counter and processor status so that program execution may continue where it was interrupted.

D. Addressing Modes

Many of the PDP-11 instructions contain one or two six bit fields for specifying an operand. The operand field consists of two three bit fields with the right three bits specifying a general register and the left three bits specifying one of eight addressing modes. (See Figure 2.2.)

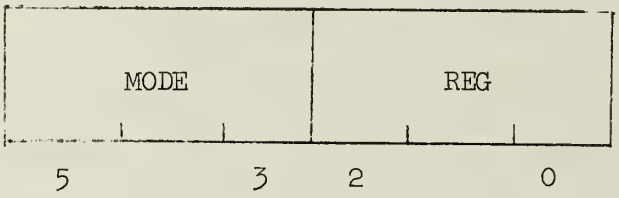


Figure 2.2 PDP-11 Operand Field

The addressing modes are:

0 - Register Mode

The operand is taken as the contents of the specified register.

1 - Register Deferred

The contents of the register are used as the address of the operand.

2 - Autoincrement

The contents of the register are used as the address of the operand. The register is then incremented by two

if the instruction is a word instruction or by one if it is a byte instruction. A special case is for byte instructions which specify R6 or R7. In these cases the register is always incremented by two.

3 - Autoincrement Deferred

The contents of the register are used to specify a location whose contents are used as the address of the operand. The register is then incremented by two.

4 - Autodecrement

The register is decremented by two for word instructions and by one for byte instructions unless the register specified was R6 or R7. In these cases it is always decremented by two. The result is then used as the address of the operand.

5 - Autodecrement Deferred

The register is decremented by two and the result is used as the address of a location whose contents is the address of the operand.

6 - Indexed

The contents of the program counter R7 are used to address a location whose contents when added to the contents of the specified register form the address of the operand. The program counter is then incremented by two.

7 - Index Deferred

The contents of the program counter R7 are used to address a location whose contents when added to the contents of

the specified register form an address whose contents are the address of the operand. The program counter is then incremented by two.

E. Instruction Format

To achieve its powerful instruction set the PDP-11 uses a variable format instruction. Seven major instruction formats are used with several others being modified versions of them. These formats are described below, and detailed descriptions of individual instructions are given in Appendix C.

The double operand format of Figure 2.3 consists of a four bit operation code, a six bit source field, and a six bit destination field. The top bit of the opcode specifies whether the instruction is to operate on an eight bit byte or on a sixteen bit word. The source and destination fields each contain a three bit register number and a three bit addressing mode as discussed previously.

The single operand instruction format shown in Figure 2.4 contains a ten bit opcode and a six bit destination field. Again the most significant bit of the opcode specifies whether the instruction is to operate on byte or word data. The destination field is the same as that for the double operand format.

The register operand instruction format given in Figure 2.5 is a modified version of the double operand format. The main difference is that one of the operands may only be register mode. A second difference is that there are no byte instructions using this format.

Branch instructions use the format shown in Figure 2.6 with a similar format being used by the TRAP and EMT instructions. The left eight bits contain the opcode and the right eight bits contain a signed offset for branch instructions and a function field for EMT and TRAP instructions.

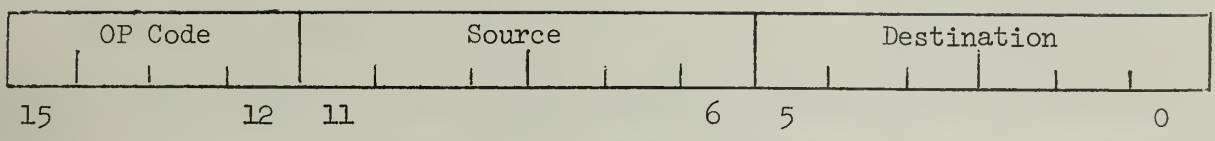


Figure 2.3 Double Operand Instruction Format

Figure 2.4 Single Operand Instruction Format

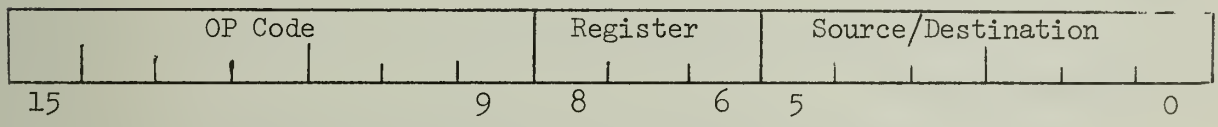


Figure 2.5 Register Operand Instruction Format

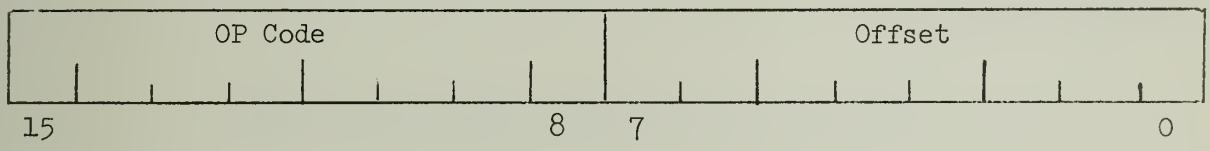


Figure 2.6 Branch Instruction Format

This function field is used by software routines and has no effect on instruction execution. In the case of the branch instructions the offset is multiplied by two and added to the program counter. This allows a forward branch of 127 words or a backward branch of 128 words.

The condition code instruction format is used only by the two condition code instructions which allow setting and clearing of condition code bits in the processor status word. As seen in Figure 2.7 the least significant four bits of the instruction specify which bits are to be set or cleared in the processor status word.

The subroutine return instruction format uses the format given in Figure 2.8, the least significant three bits specifying the register to be used for the return. The remaining instructions use only an opcode as shown in Figure 2.9.

F. New Instructions

Although the PDP-11 instruction set is very flexible it was felt that several new instructions were desirable. These instructions require three operands requiring the addition of another instruction format. The format is shown in Figure 2.10 and the new instructions are described in detail in Appendix C.

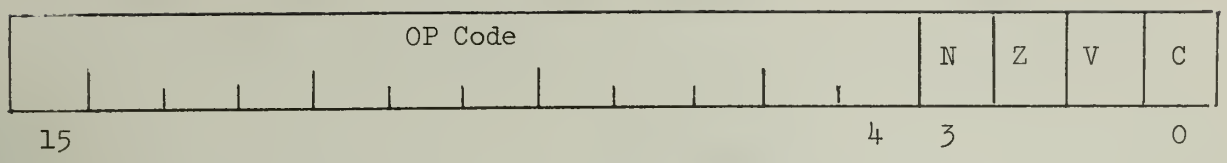


Figure 2.7 Condition Code Instruction Format

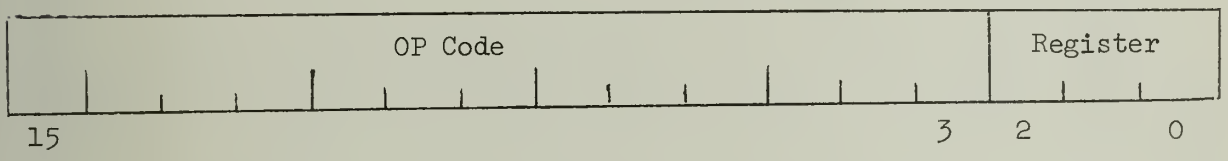


Figure 2.8 Subroutine Return Instruction Format

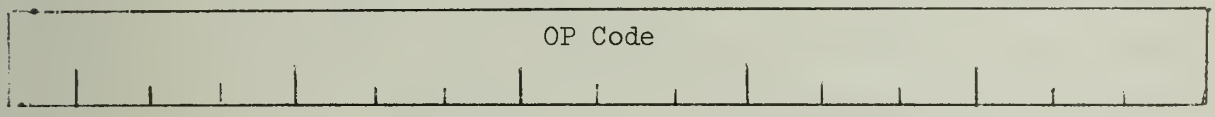


Figure 2.9 Miscellaneous Instruction Format

Figure 2.10 Triple Operand Register Format

CHAPTER III

THE EMULATOR

This chapter describes the PDP-11 emulator written for the SUE. The emulator includes all of the PDP-11/40 instructions and four additional instructions as well as all of the necessary firmware to support a memory mapping unit should one be built for the SUE in the future. The full emulator requires 512 words of table space and 1024 words of control store. A smaller version can also be used which excludes the MTPI, MFPI, MUL, DIV, ASH, ASHC, STM, LDM, BTS and BTM instructions. This version requires only 768 words of control store.

The emulator executes instructions at approximately the same speed as the PDP-11/20. Instructions using register mode are slightly slower than those of the 11/20 while many instructions using other modes are faster. A complete listing of instruction execution times are given in Appendix A.

A. Register Allocation

The SUE register file contains twelve general purpose registers. The first eight of these registers are used to store the PDP-11 registers. Register six is the stack pointer and register seven the program counter.

Register eight is used to store the processor status word. This register is unique in that the four most significant bits are used to mask off I/O interrupts. Register nine is the instruction register. It contains the instruction currently being executed. The remaining two registers

are used by the emulator for temporary storage and their contents depend on the instruction being executed and the state of the execution.

B. Processor Status Word

A description of the PDP-11 processor status word format was given in Chapter II. Because of hardware differences between the SUE and the PDP-11 it was necessary to store the processor status word internally in a format that is different from the PDP-11. The internal format is shown in Figure 3.1. The four most significant bits contain an interrupt mask. This mask is transferred to the bus controller each time register eight is loaded. This is a hardware function and as a result these bits cannot be used for any other function. As used by the emulator the mask corresponds to the priority field as given in Table 3.1.

	Priority	Interrupt Mask
Bits	6 5 4	15 14 13 12
	0 0 0	0 0 0 0
	0 0 1	0 0 0 0
	0 1 0	0 0 0 0
	0 1 1	0 0 0 0
	1 0 0	0 0 0 1
	1 0 1	0 0 1 1
	1 1 0	0 1 1 1
	1 1 1	1 1 1 1

Table 3.1 Priority Interrupt Mask Encoding

Since the processor status word can be accessed directly as an address and also saved in memory during interrupts the processor status word must be reformatted each time it is accessed. The fact that an instruction can

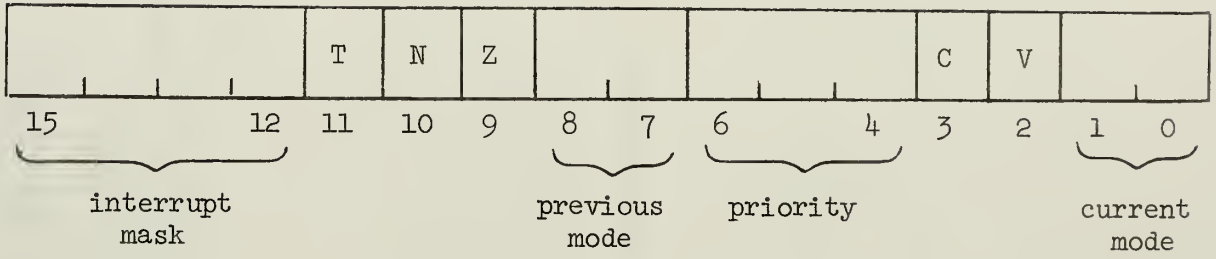


Figure 3.1 Internal Processor Status Word Format

access the processor status as an address creates a problem for the SUE because no hardware is provided to detect when the processor status word is being accessed, and the overhead required to check each address in the firmware would be prohibitive. Instead the firmware attempts to access the processor status word as Infibus address 177776_8. Since this address in nonexistant a timeout will occur on the bus after two microseconds. At this point a check is made in the firmware to determine if the program was trying to access the processor status. If it was, the status is fetched from register eight, converted from internal format to PDP-11 format and used as the data. This makes access to the status word very slow. However, the status word is seldom accessed as an address. A similar process is used when an attempt is made to write into the status word.

C. Memory Mapping and Protection Features

The memory mapping features make use of the two additional address lines provided by the SUE. These address lines are normally set to the current mode as specified in the processor status. These lines can be used by a memory mapping device to select a set of registers which would be used for mapping a virtual address given by address lines 0 through 16 into a physical address on the bus. Such a unit would be placed on the bus between the SUE processor and its memory and peripherals.

To prevent a normal user from simply changing bits to get into another mode the mode bits are always ORed into the status word whenever it is loaded via an RTI or RTT instruction. This allows each successively lower mode to have control over all modes higher than it. Mode zero is kernal mode, and it has control over all other modes. It is the only mode in which a program can address the status word directly and also the only

mode in which the HALT and SPL instructions can be executed. In other modes the HALT instruction will cause an illegal instruction interrupt to occur, and the SPL instruction will be executed as a no-op.

Interrupts must be handled differently than on the PDP-11/40 because the emulator has only one stack register. The PDP-11/40 pushes the old program counter and status word on the new modes stack. The emulator is forced to push them on the old modes stack since each mode does not have its own stack register. The software must then change the stack pointer and retrieve the old program counter and status word and move it onto its own stack.

D. Addressing Modes

The emulator contains seven sets of addressing modes. They are required because no microsubroutine linkage is provided in the SUE hardware. There is a set of mode routines for each of the following:

single operand word instructions

single operand byte instructions

double operand word instructions - source mode

double operand byte instructions - source mode

double operand word instructions - destination mode

double operand byte instructions - destination mode

single and register operand special instructions

The register operand and single operand special instructions include the JSR, JMP, SXT, MTPI, MFPI, MUL, and DIV. The mode routines for these instructions differ from the others in that only the address of the operand is calculated. In all other cases the operand is actually fetched.

E. Instruction Decoding

Instruction decoding is performed using the microlevel table. A particular row of the table is selected, and a four bit field of the instruction is used to index into that row. The element obtained from the table is then used to either select another row to be used to access the table again in the next microinstruction, or it can be used as a jump address to which the next microinstruction jumps. Because most fields in the PDP-11 instruction set are only three bits wide and are not aligned on a four bit boundary the instruction must often be shifted to align the fields for decoding. Also as a result of the three bit fields many entries within a row in the table are repeated.

Decoding begins with the four most significant bits of the instruction. These bits are used to access one of sixteen elements in row seven of the table. The element selected specifies the next row to be used for decoding bits eight through eleven. Decoding of bits twelve through fifteen results in the instructions being dividied into six groups as shown in Figure 3.2. The floating point group is unimplemented and results in the selection of a row in which all of the entries are addresses of the illegal instruction routine. The other groups contain the following instructions:

Group 0

CLR, DEC, INC, NEG, TST, COM, ASL, ASR, ADC, SBC,
SXT, ROL, ROR, SWAB, BR, BEQ, BNE, BLT, BGE, BLE,
BGT, JSR, MARK, RTS, SPL, JMP, BPT, IOT, BTS, RTT,
RTI, HALT, WAIT, RESET, MTPI, MFPI, and the
condition code instructions.

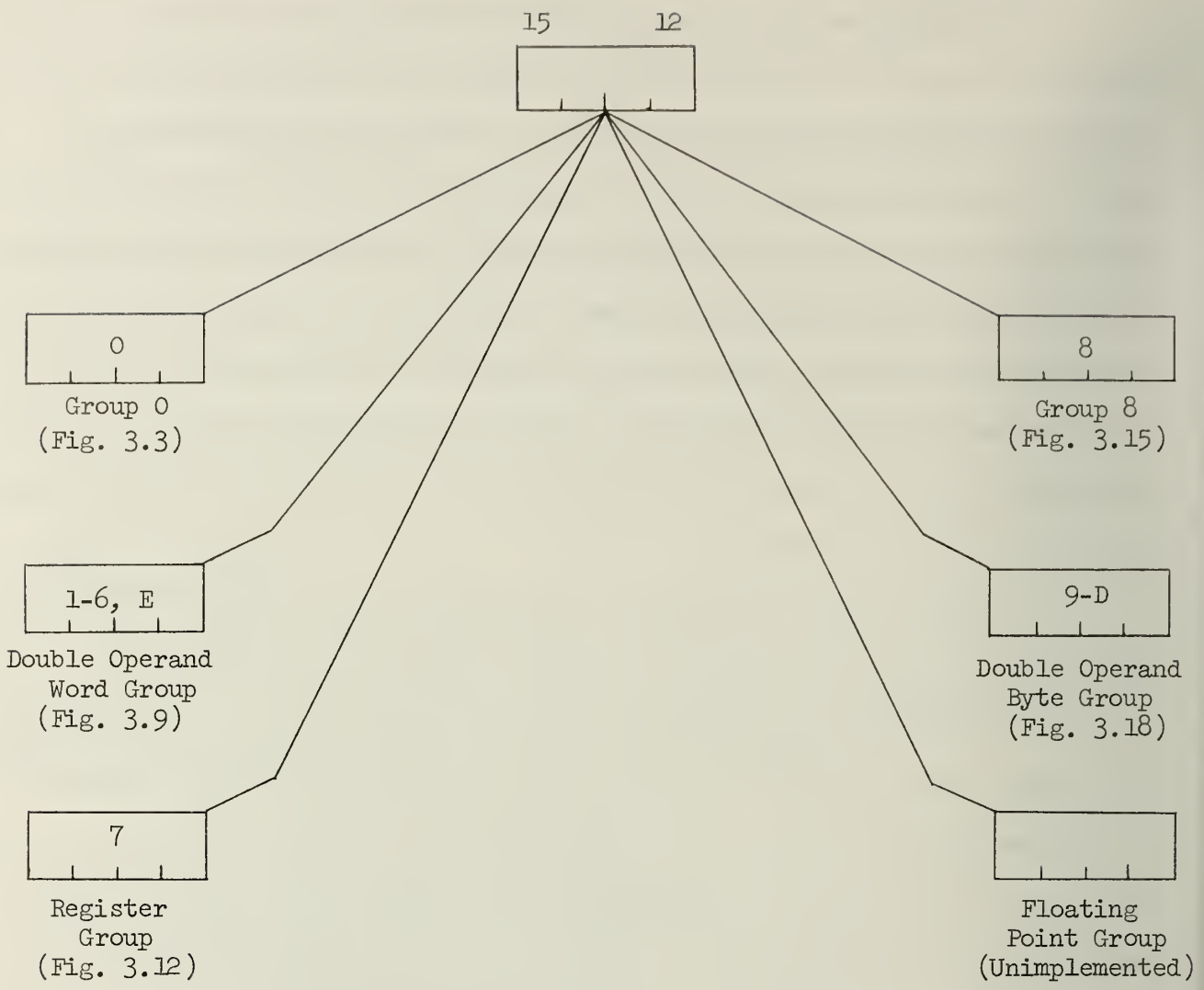


Figure 3.2 Instruction Decoding of
Bits 12 - 15

Double operand word group

MOV, CMP, BIT, BIC, BIS, ADD, and SUB

Group 8

CLRB, DECB, INCB, NEGB, TSTB, COMB, ASLB, ASRB,

ADCB, SBCB, ROLB, RORB, BMI, BPL, BCS, BCC, BVS,

BVC, BHI, BLOS, BLO, BHIS, EMT, and TRAP

Double operand byte group

MOVB, CMPB, BITB, BICB, and BISB

Group 0 decoding

Decoding of bits eight through eleven of group 0 separates the individual branch instructions within the group as well as the JSR and BTS instructions. This is diagrammed in Figure 3.3. The remaining instructions are divided into three groups containing the following instructions:

Group 0.0

HALT, WAIT, RTI, BPT, IOT, RESET, RTT, RTS, SPL,

SWAB, JMP, and the condition code instructions.

Single operand word group

CLR, DEC, INC, NEG, TST, COM, ASL, ASR, ADC, SBC,

ROR, and ROL

Single operand special group

MARK, MFPI, MTPI, and SXT

The branch instructions need no further decoding so control passes directly to the execution routines. The BTS instruction likewise can be executed without further decoding. The JSR and single operand word group instructions require decoding of their addressing mode. The single operand special instructions also require address mode decoding as well as other decoding.

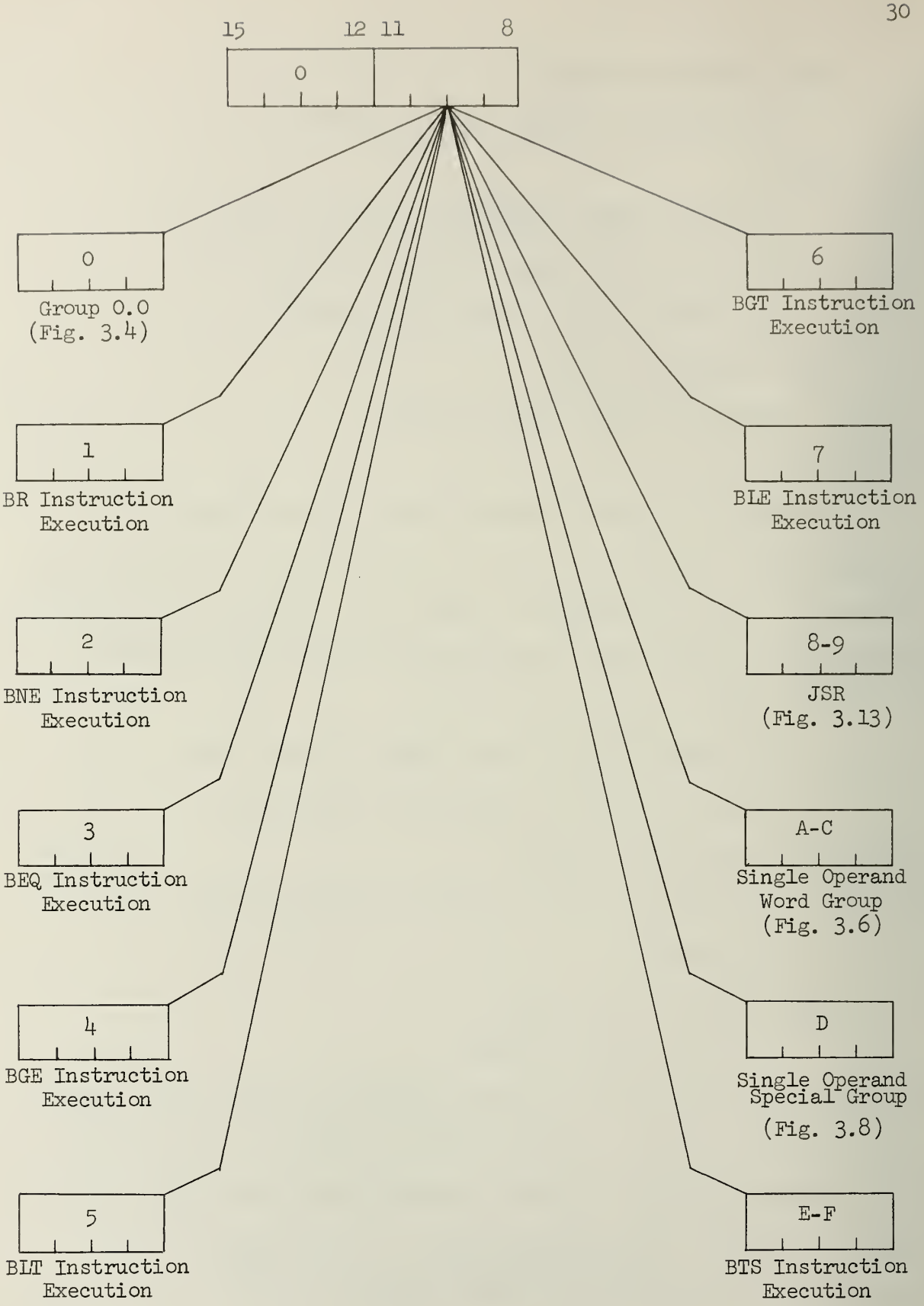


Figure 3.3 Group 0 Decoding,
Bits 8 - 11

Group 0.0 decoding

Bits four through seven are used to further decode the instructions in this group as illustrated in Figure 3.4. Individual instructions separated are JMP, RTS, SPL, SWAB, and the condition code instructions. The RTS, SPL, and the condition code instructions are executed without further decoding. The JMP and SWAB must have the addressing mode decoded. The SWAB instruction is a normal single operand instruction and is treated as such in any further decoding. However, before control is passed to one of the single operand mode routines the SWAB instruction's opcode is modified by setting bit eight in the instruction. This allows easier decoding into individual instructions after exiting the mode routines. The JMP instruction uses the single operand special mode routines, and like the SWAB instruction the opcode for JMP is also modified. In this case it is modified by setting bit twelve.

The remaining instructions in group 0.0 are either illegal opcodes or are part of group 0.0.0. Decoding of this group is shown in Figure 3.5. The group includes the HALT, WAIT, RTI, BPT, IOT, RESET, and RTT instructions as well as nine illegal opcodes.

Single operand word group decoding

Further decoding of the single operand word instructions is given in Figures 3.6 and 3.7. Only five of the addressing mode routines can be entered directly. The other three modes differ only in that they are indirect modes and require only an additional memory fetch. Decoding into individual instructions is performed using bits six through nine as shown in Figure 3.7. The decoding of the SWAB instruction is the result of the modification to the opcoce as discussed previously.

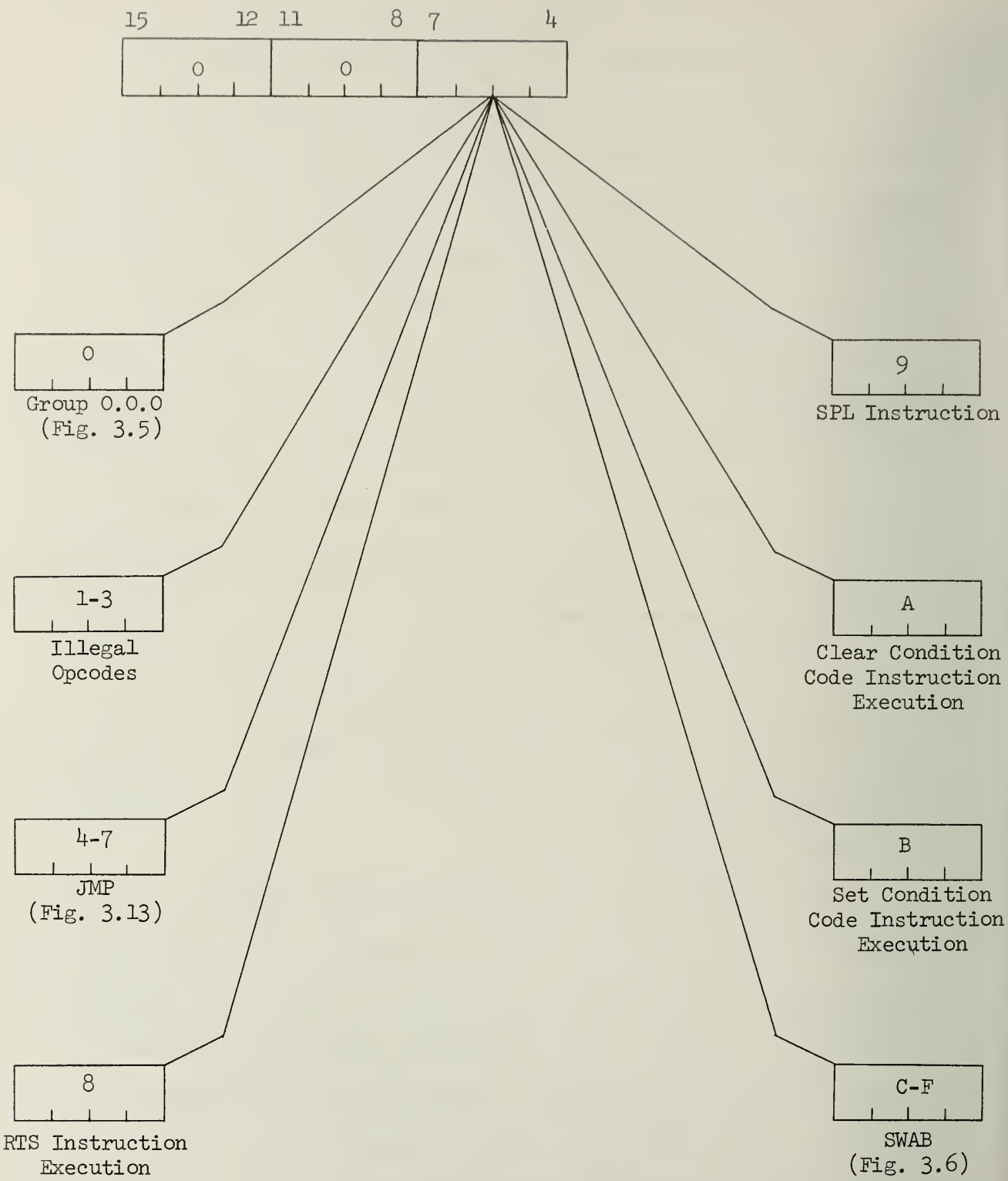


Figure 3.4 Group 0.0 Decoding,
Bits 4 - 7

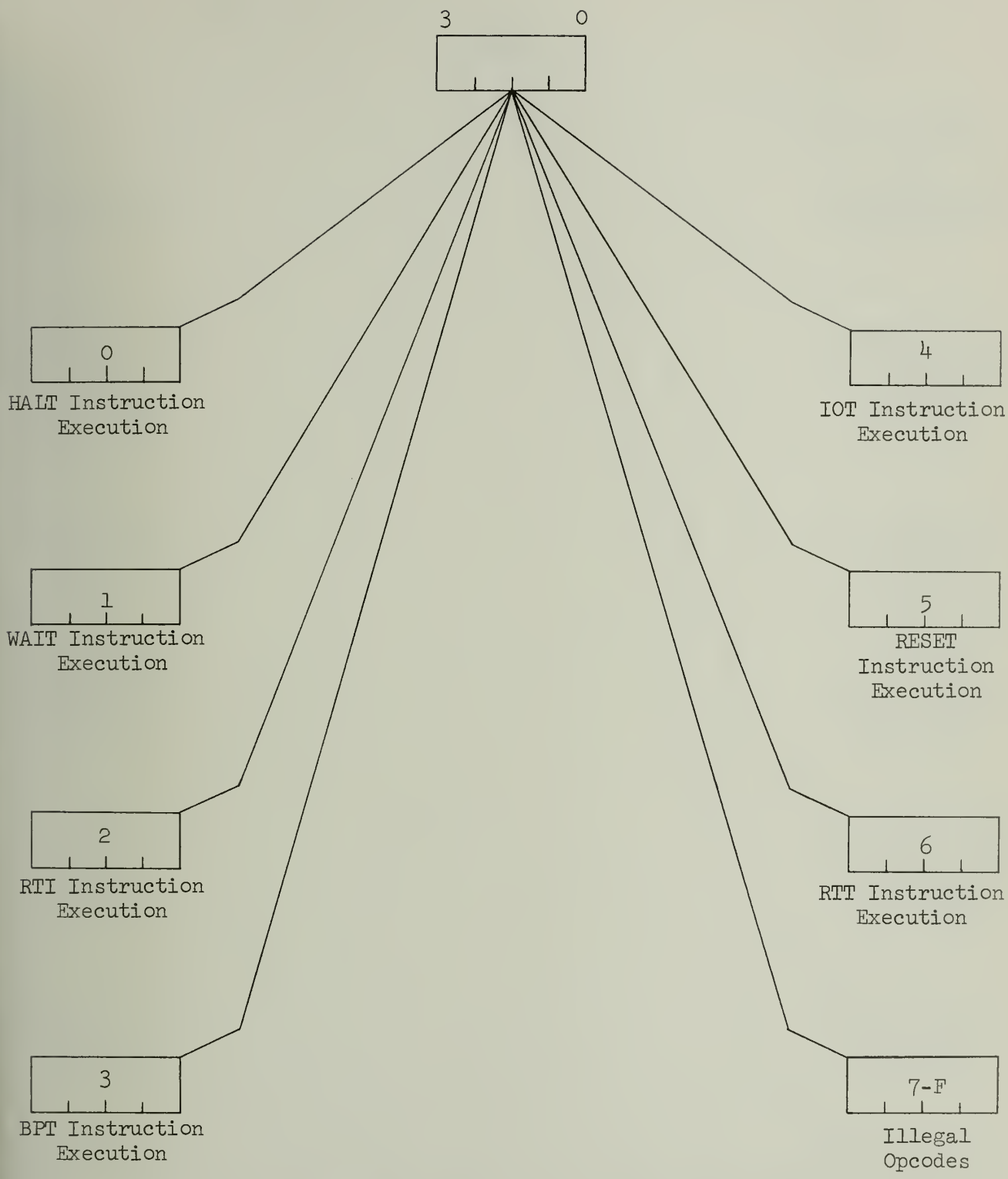


Figure 3.5 Group 0.0.0 Decoding, Bits 0 - 3

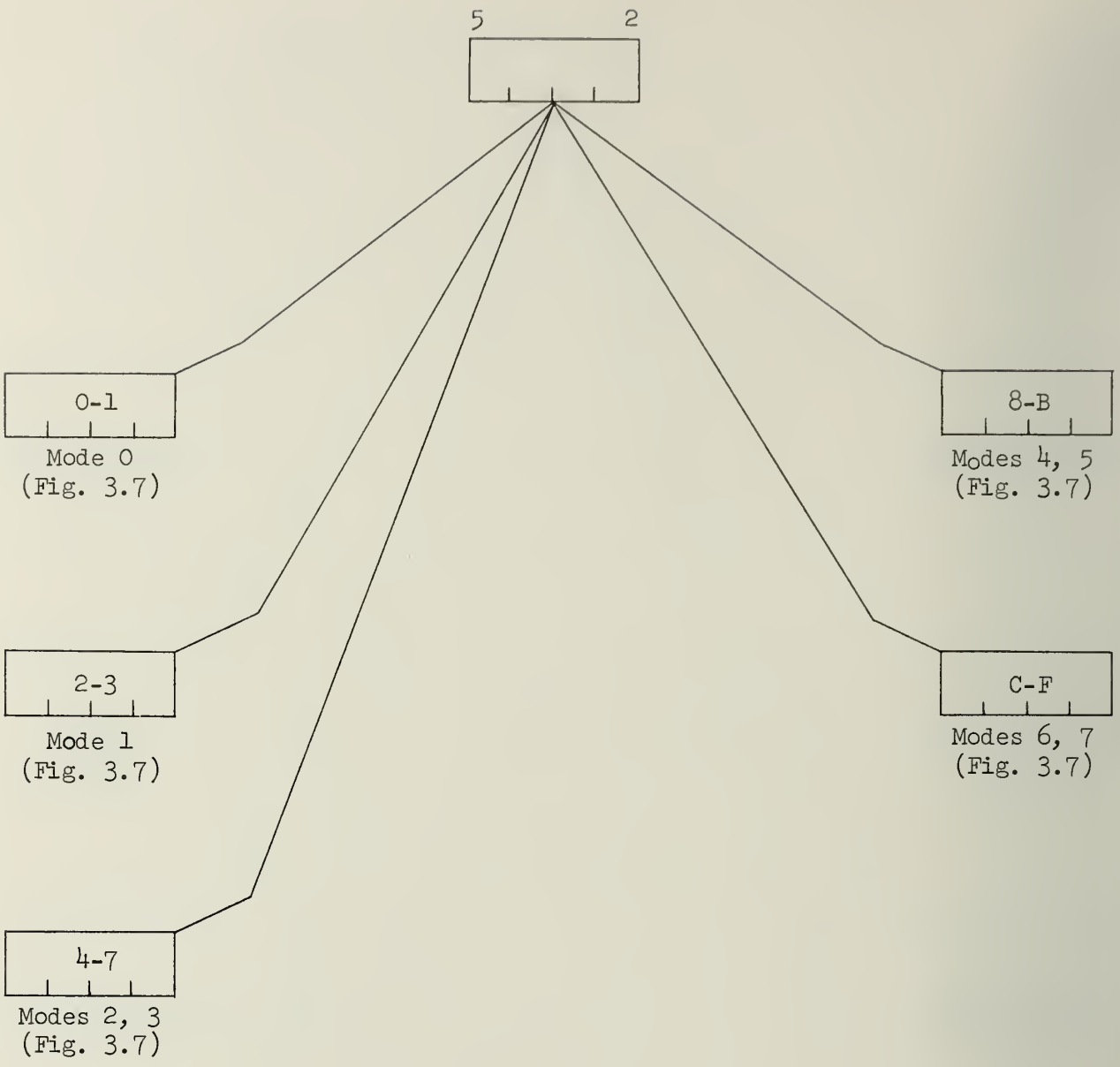


Figure 3.6 Single Operand Word Instruction
Mode Decode

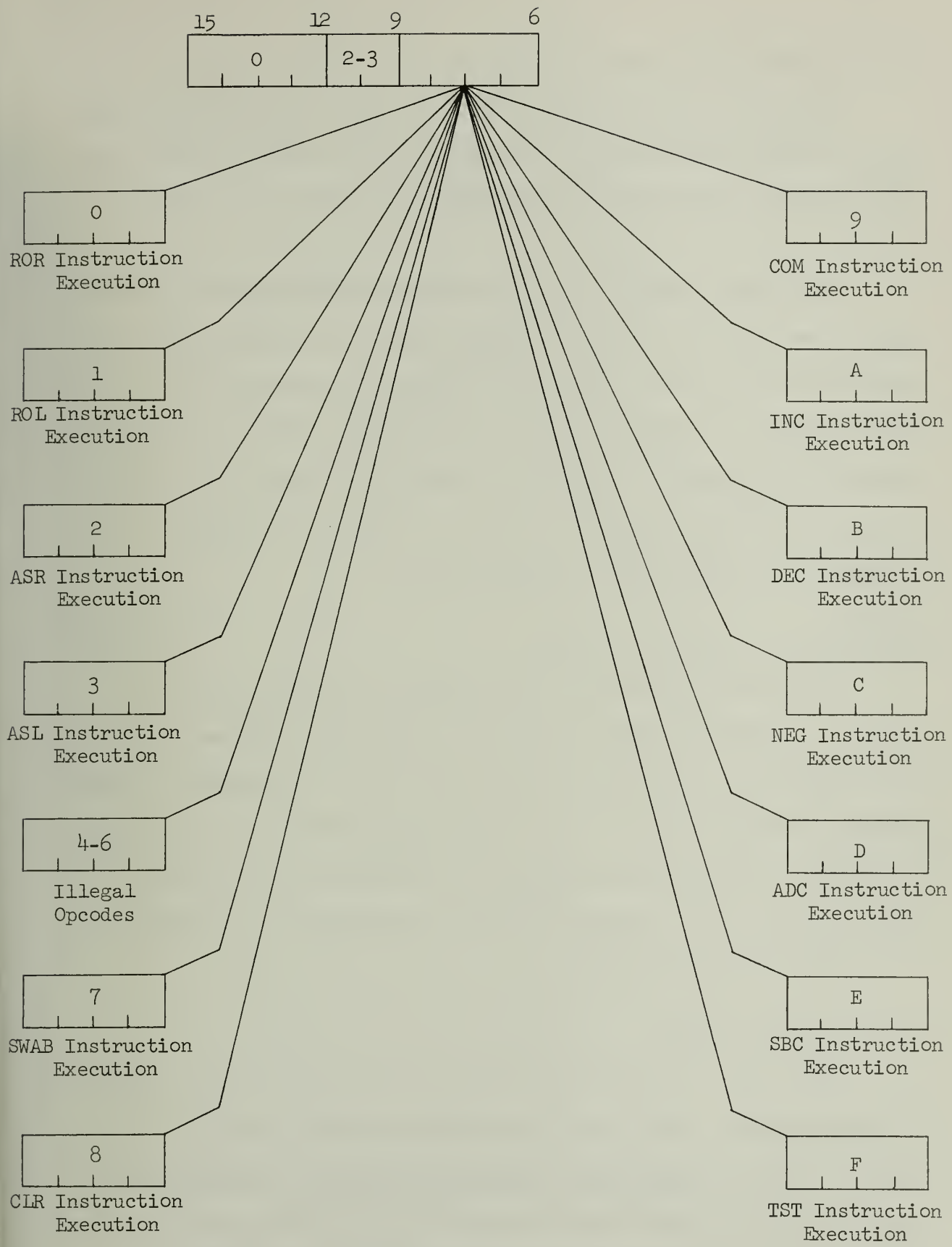


Figure 3.7 Single Operand Word Instruction Decoding

Single operand special group decoding

Figure 3.8 shows the decoding of the single operand special group. Bits six through nine are used to select the table element, however, since bits eight and nine are fixed the decoding is actually only on bits six and seven.

The MARK is executed with no further decoding while the other three instructions require decoding of the addressing mode. Again opcodes are modified to allow easier decoding into individual instructions later. The SXT instruction is modified by setting bit fourteen, MFPI by setting bits fourteen and fifteen, and the MTPI by setting bits thirteen and fourteen. Decoding of the addressing mode is shown in Figure 3.13. Further decoding of these instructions will be discussed later.

Double operand word group decoding

The double operand word group decoding is given in Figures 3.9 through 3.11. Double operand instructions require the decoding of two sets of addressing modes. Like the single operand word instructions, the mode routines for modes 3, 5, and 7 are combined with modes 2, 4, and 6 respectively. Decoding into individual instructions for execution is given in Figure 3.11. Both word and byte instructions use the same row in the table for this decoding.

Register group decoding

The register group of instructions is decoded as shown in Figure 3.12. The ASH, ASCH, STM, LDM, and SOB instructions are fully decoded and are executed. The XOR instruction is treated identically to a double operand instruction with a source addressing mode of 0. Thus upon detection of an XOR instruction control is passed directly to the double operand word

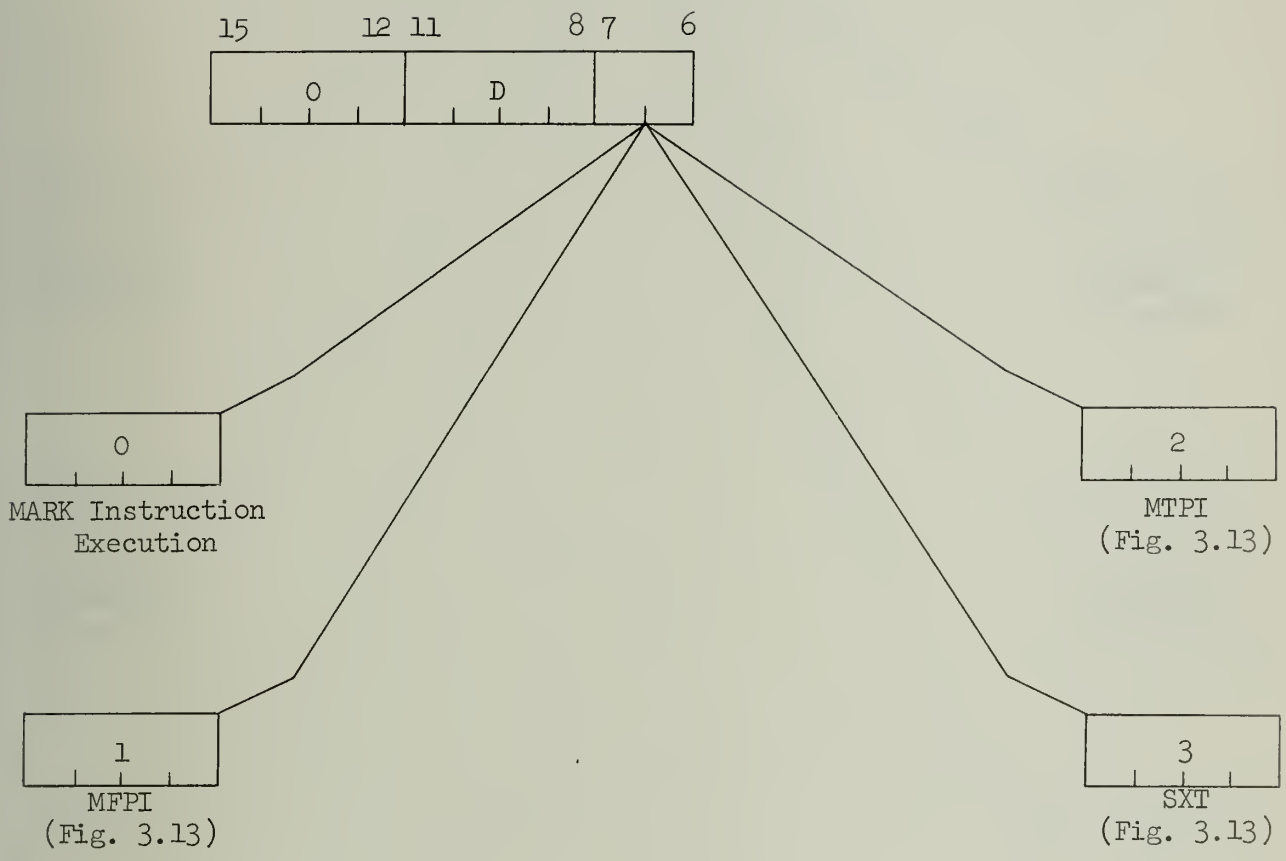


Figure 3.8 Single Operand Special Instruction Decoding

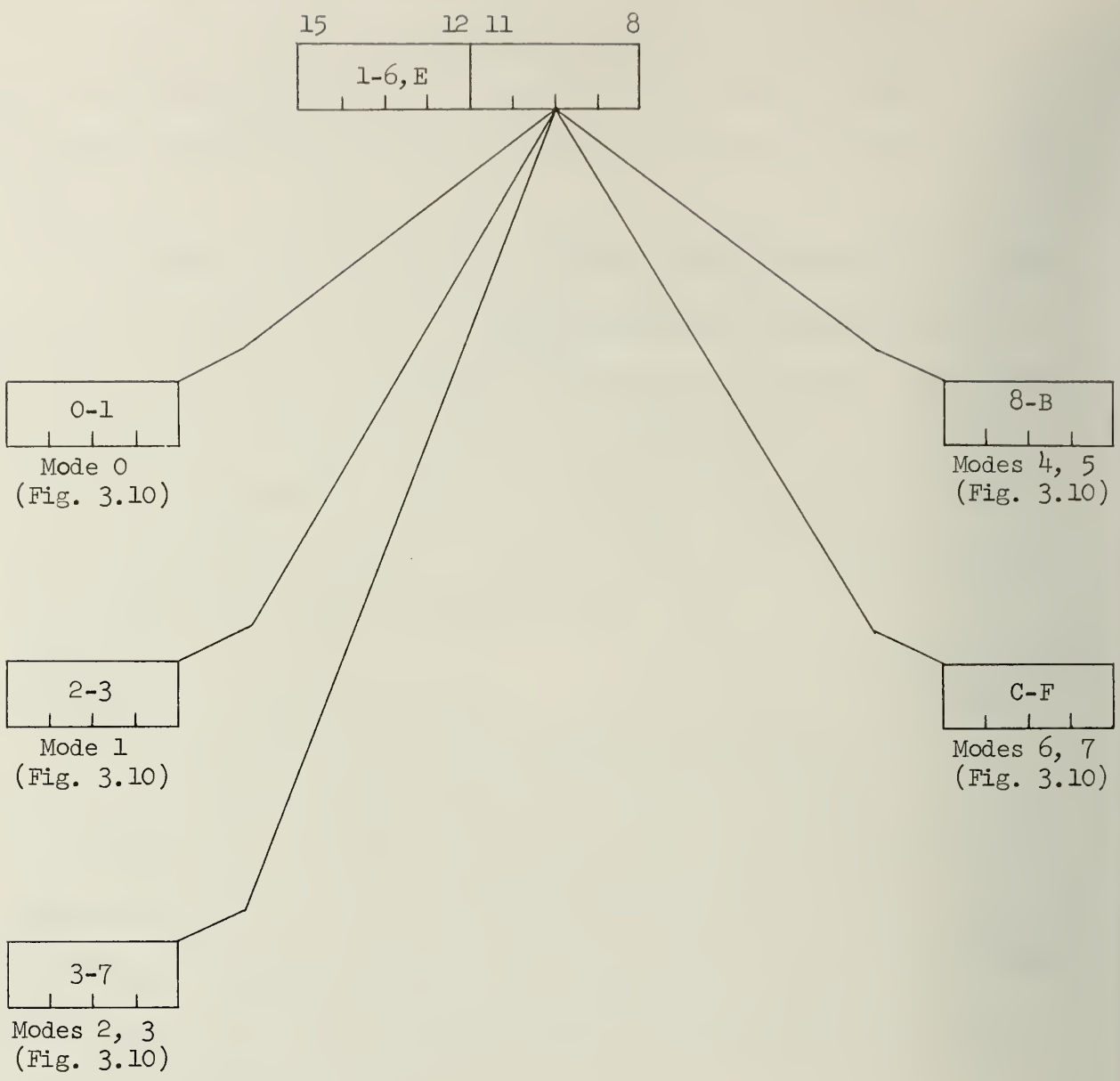


Figure 3.9 Double Operand Word Instruction
Source Mode Decoding

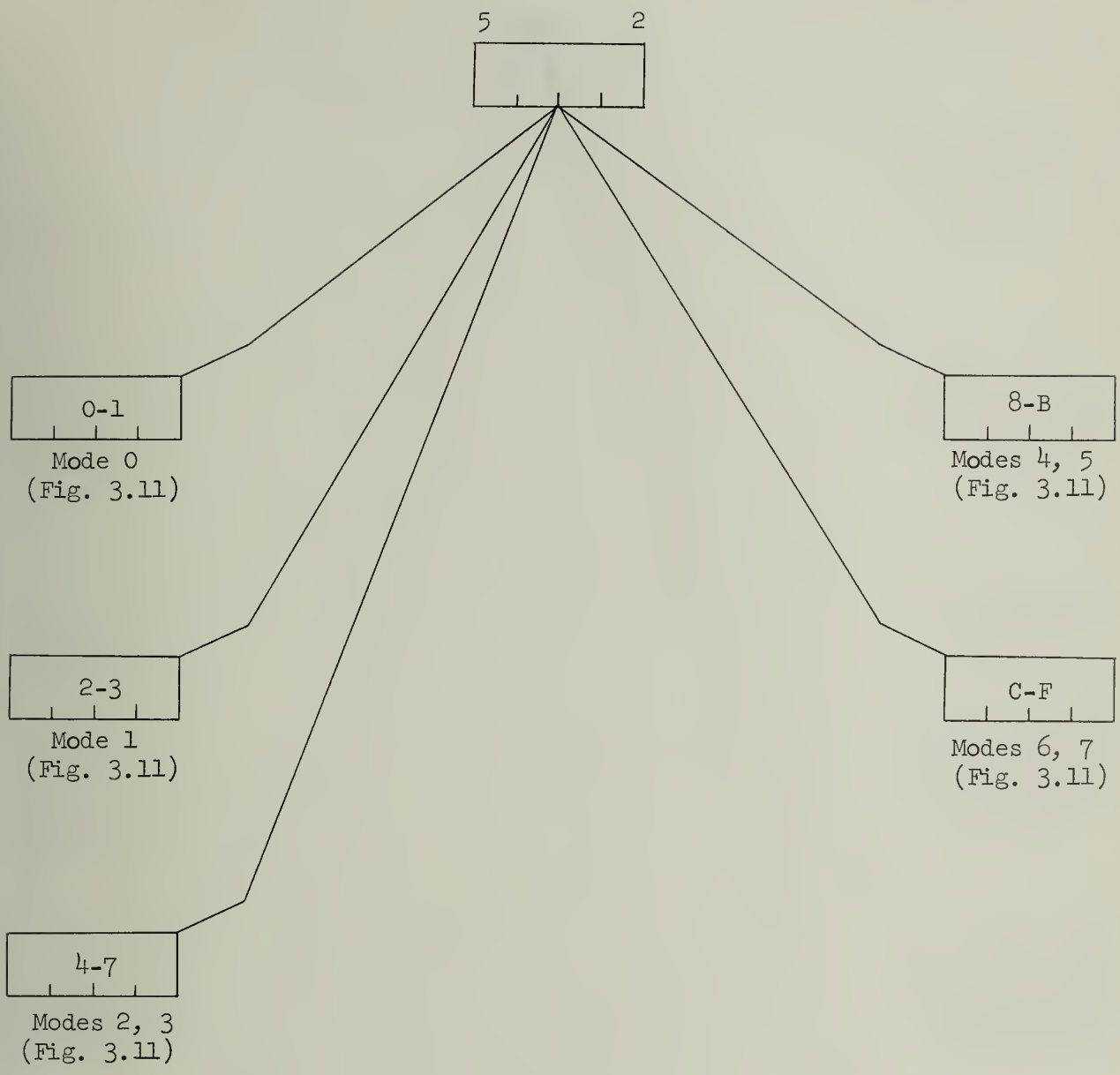


Figure 3.10 Double Operand Word Instruction
Destination Mode Decoding

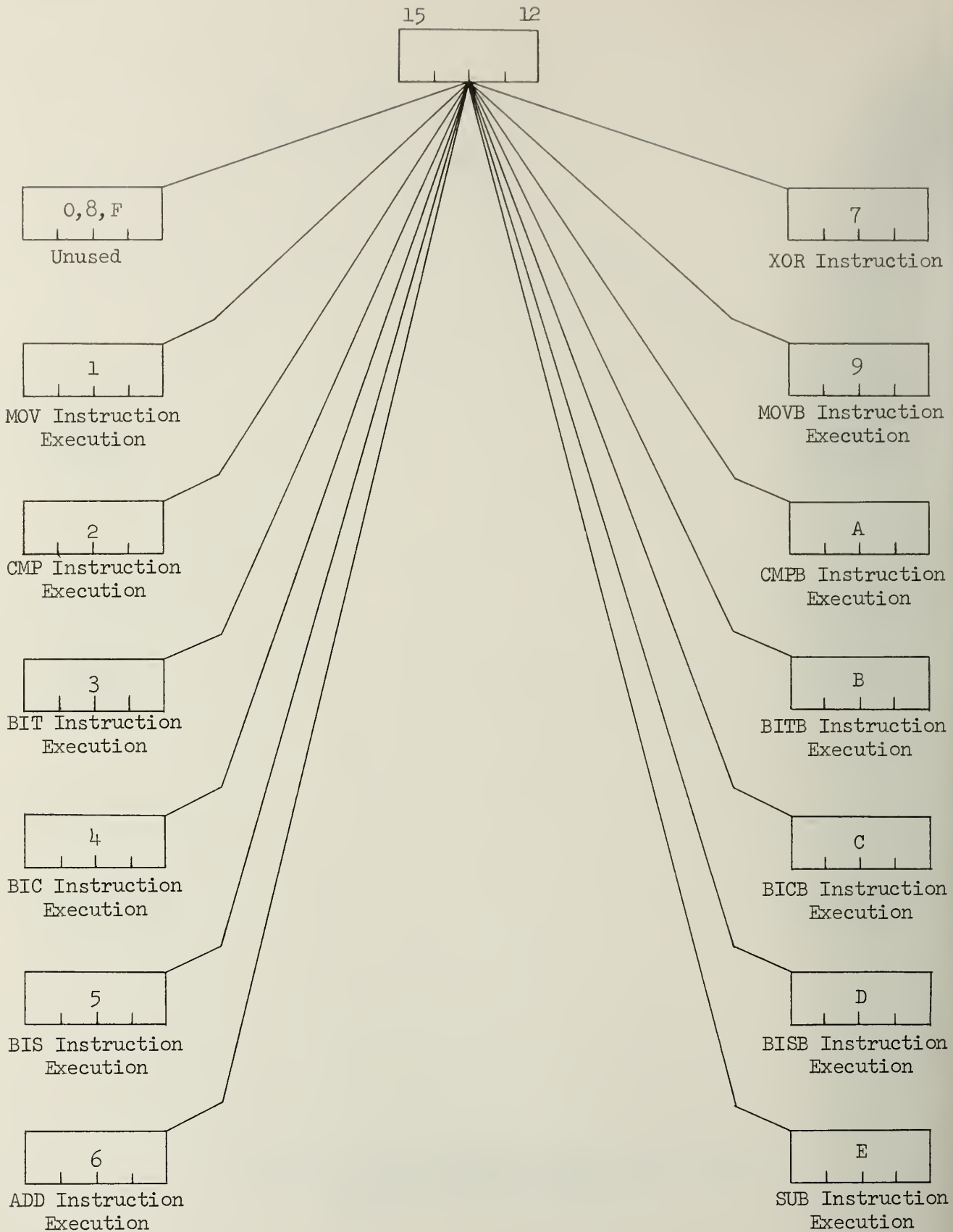


Figure 3.11 Double Operand Instruction Decoding

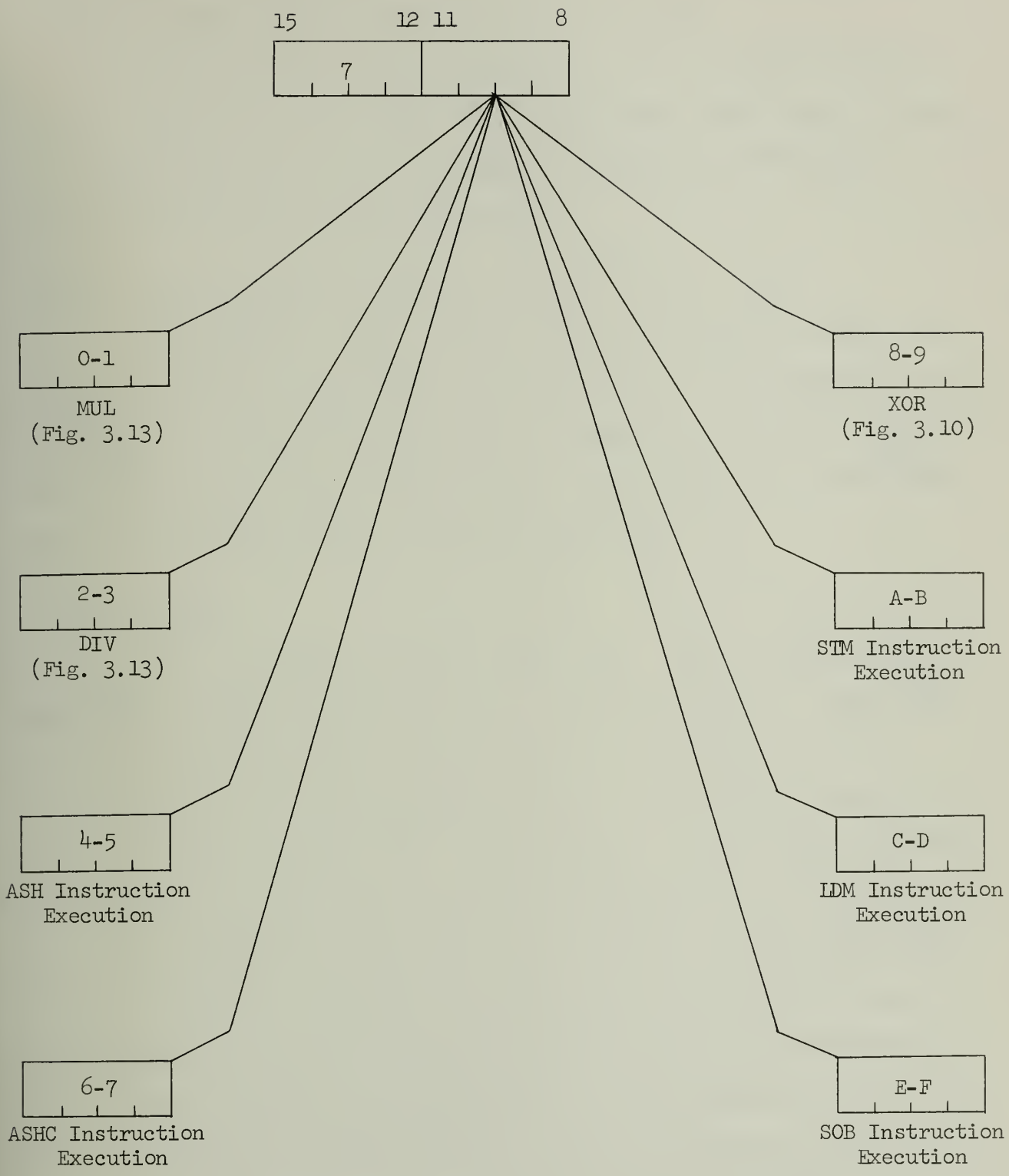


Figure 3.12 Register Group Decoding

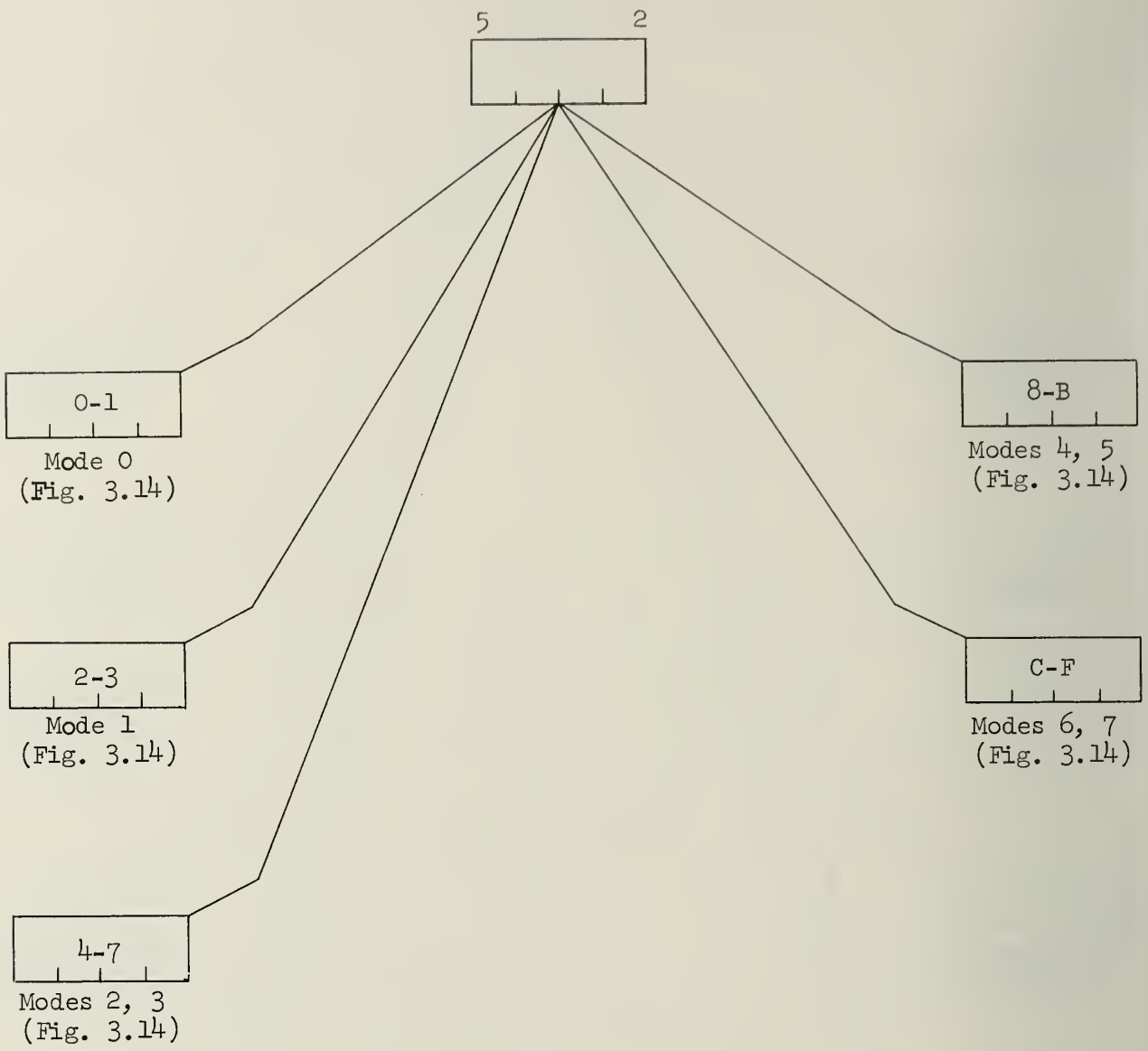


Figure 3.13 Single Operand Special Instructions
Mode Decoding

source mode routine, and any further decoding of the XOR instruction is performed as though it were a double operand instruction.

The MUL and DIV instructions require decoding of their addressing modes. This is done as shown in Figure 3.13. Before control passes to one of the mode routines the opcode of the MUL instruction is modified by clearing bit thirteen of the instruction.

Special single operand and register group instruction decoding

Seven instructions use the single and register operand special mode routines. These include the JSR, JMP, SXT, MUL, DIV, MTPI, and MFPI instructions. Each of these instruction's opcode was modified if necessary so that the four most significant bits are unique. This results in decoding into individual instructions as shown in Figure 3.14.

Group 8 decoding

Group 8 decoding is given in Figure 3.15. After decoding of bits eight through eleven the remaining branch instructions as well as the EMT, TRAP, and BTM instructions are fully decoded. The single operand byte instructions are the only ones requiring further decoding. The modes are decoded as shown in Figure 3.16. Unlike the mode routines for the word instructions only modes six and seven can be combined. This is because the autoincrements and autodecrements vary depending on whether the mode is direct or indirect. An indirect mode always results in an autoincrement or autodecrement by two while the direct mode may result in an autoincrement or decrement by either one or two depending on which register is being used.

Decoding into individual instructions is illustrated in Figure 3.17.

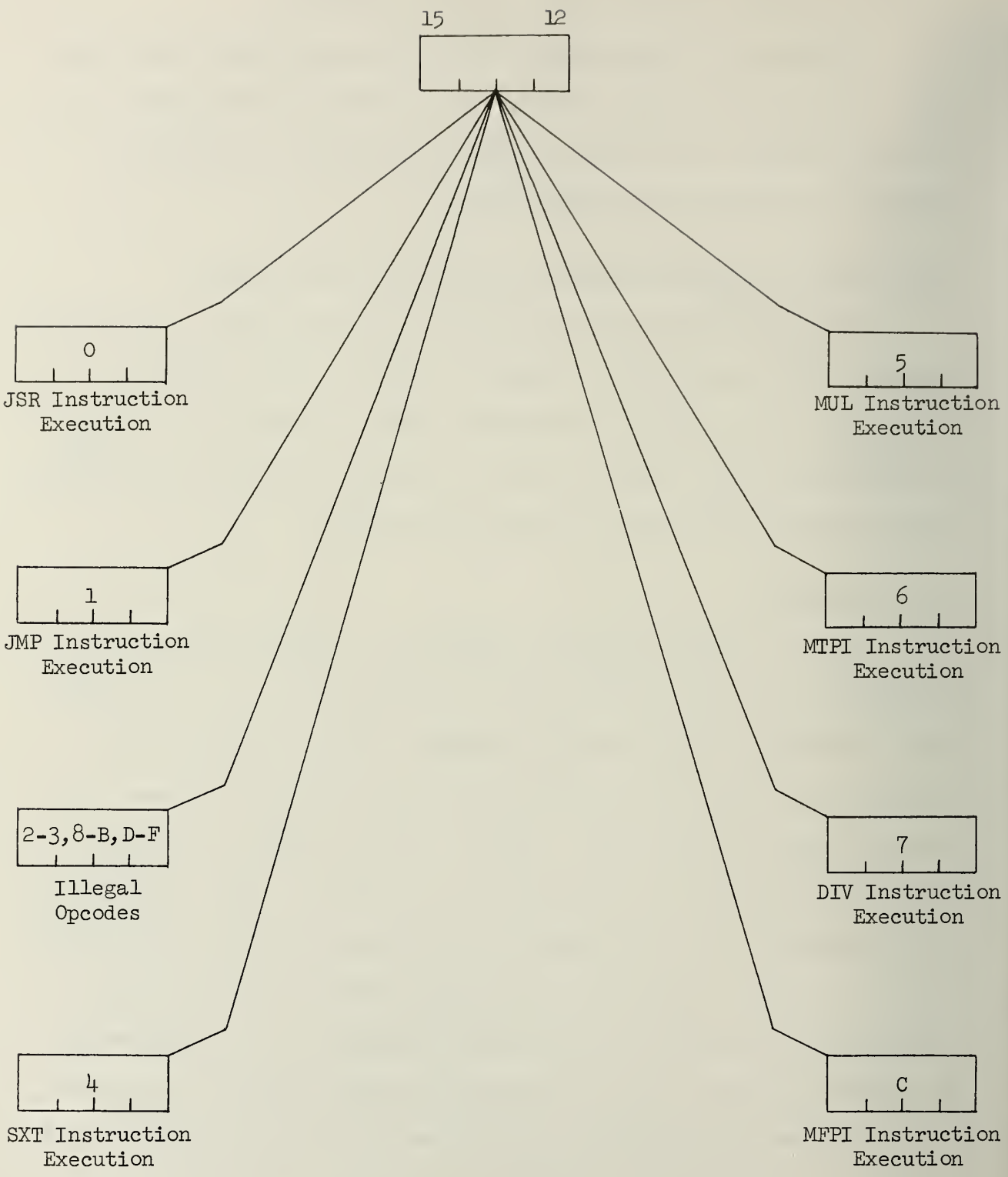


Figure 3.14 Special Single Operand and Register Group Instruction Decoding

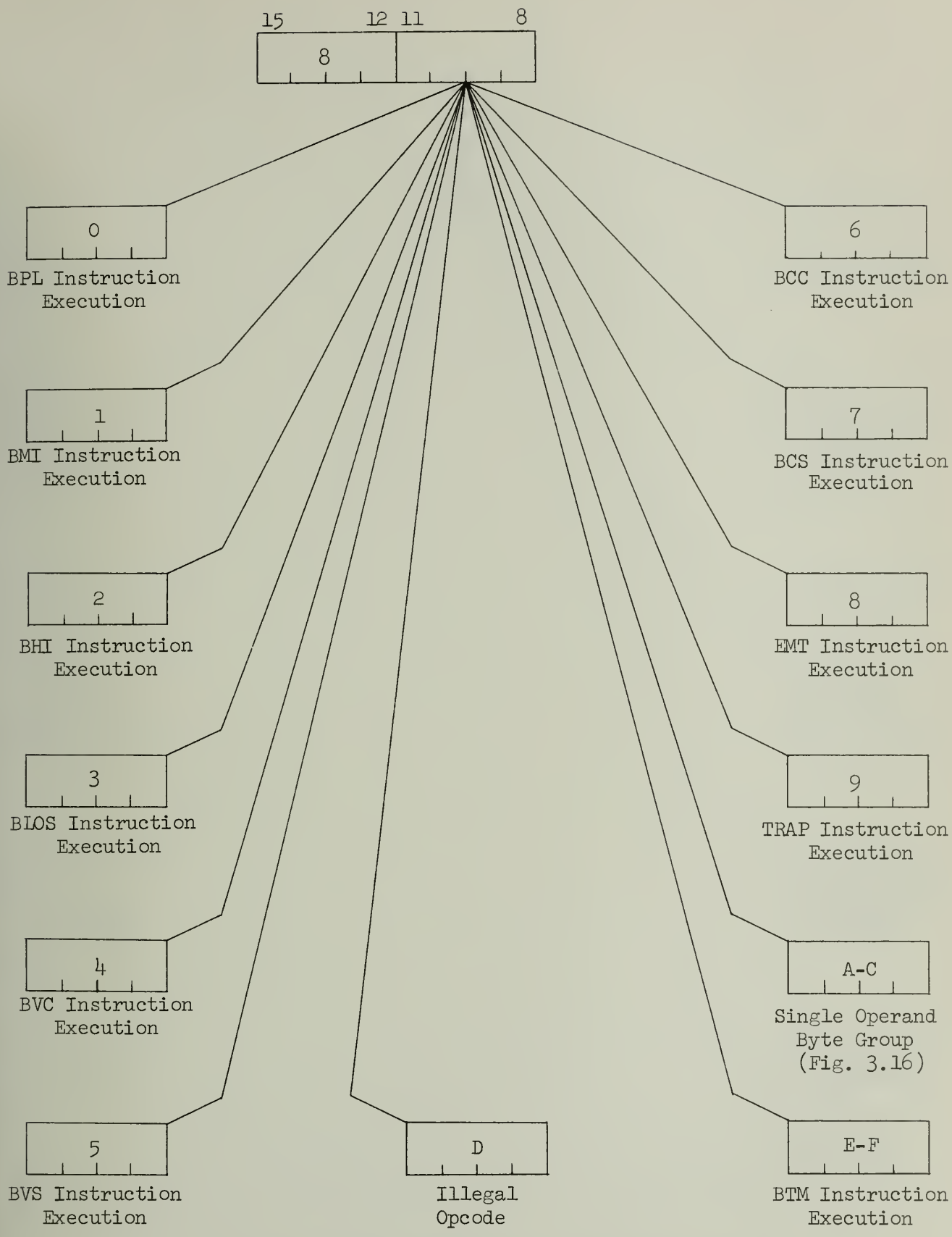


Figure 3.15 Group 8 Decoding, Bits 8 - 11

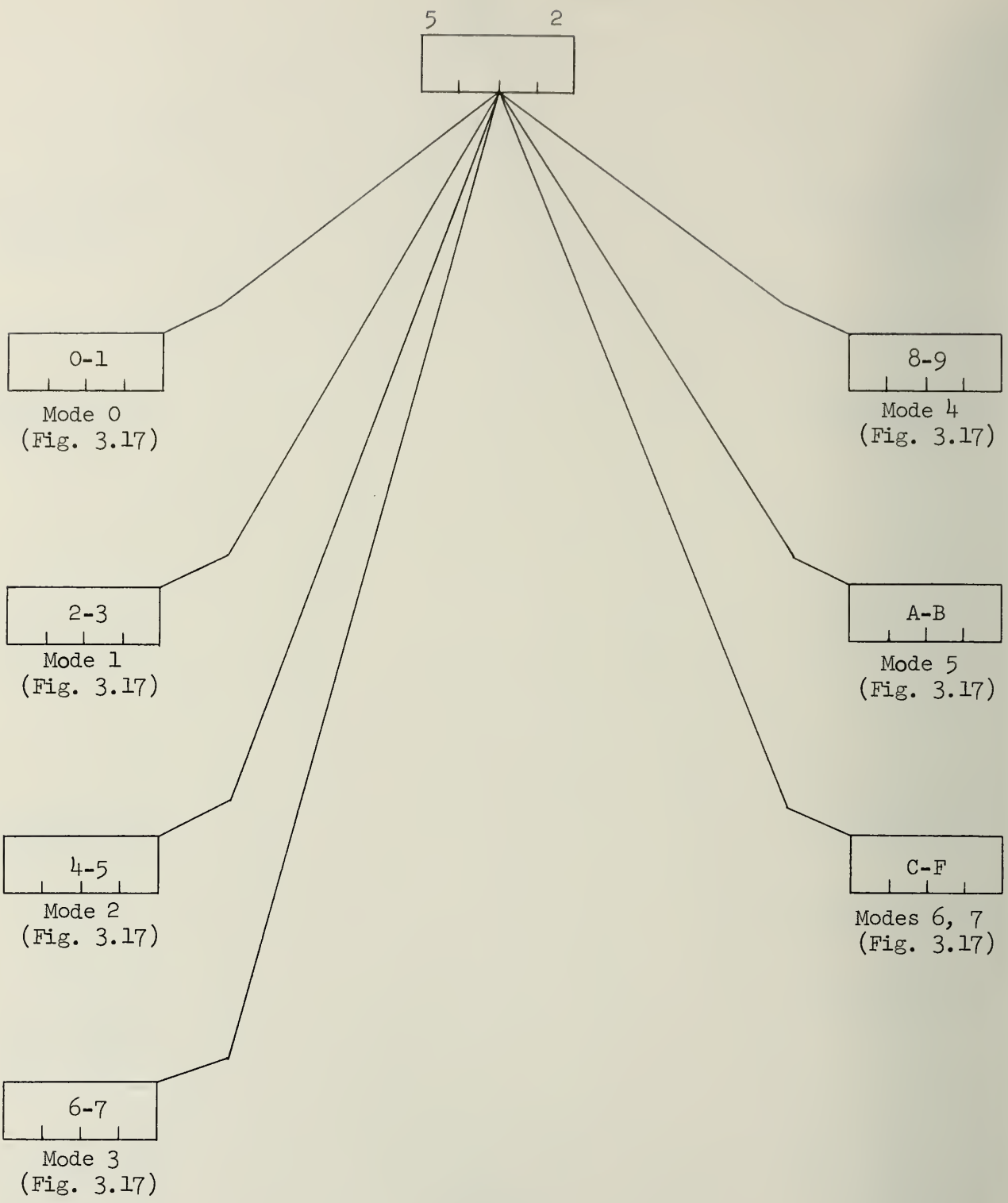


Figure 3.16 Single Operand Byte Instruction
Mode Decoding

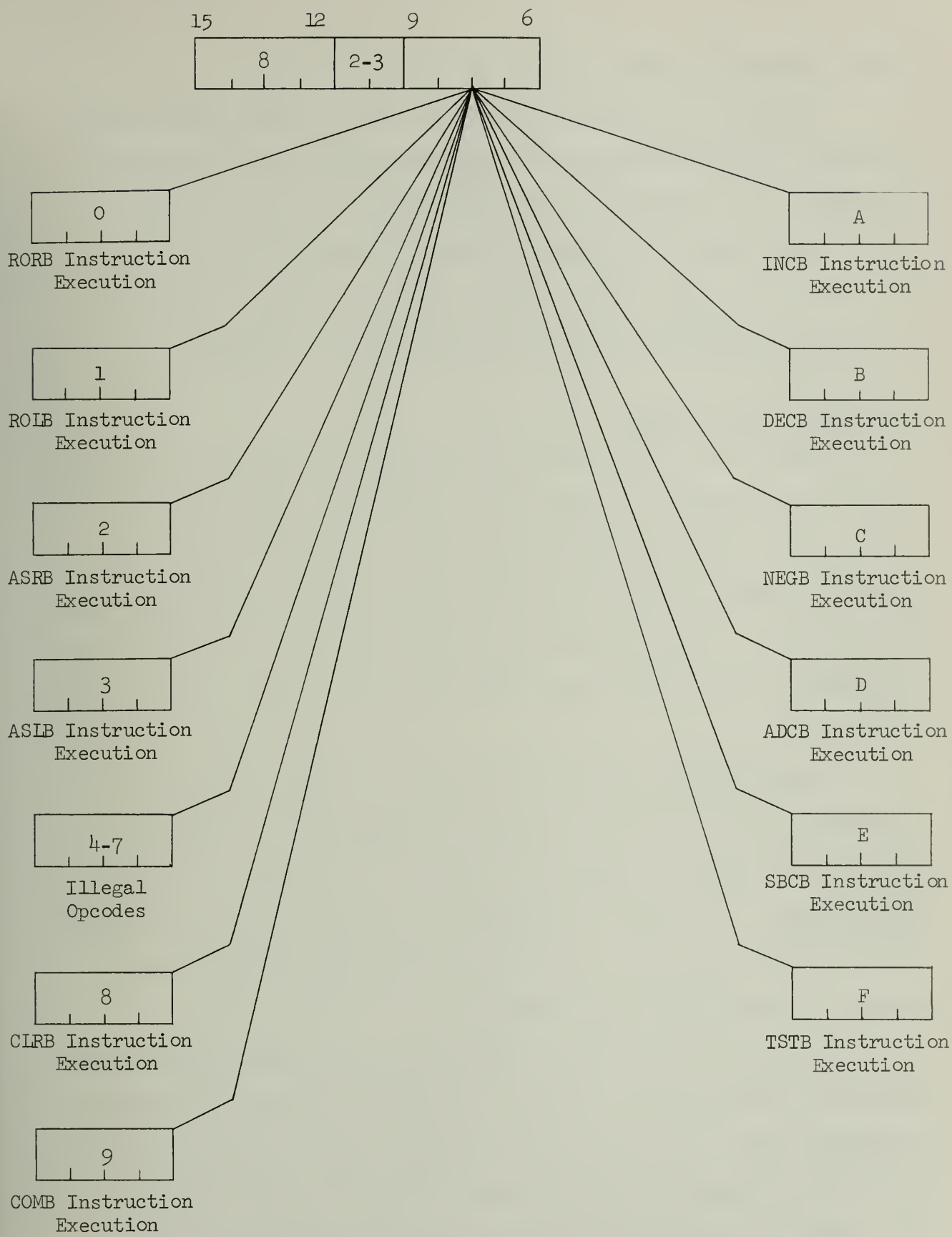


Figure 3.17 Single Operand Byte Instruction Decoding

Double operand byte group decoding

Decoding of the double operand byte instructions continues with the decoding of the source mode as shown in Figure 3.18 and the destination mode in Figure 3.19. For the same reason as the single operand byte modes, the double operand byte modes require a separate routine for each mode with the exception of modes six and seven which can be combined. Separation into individual instructions for execution is given in Figure 3.11.

F. Instruction Execution

Execution of those instructions which do not modify the condition codes is straightforward. Those that do change the condition codes however, require considerable manipulation and testing to determine and set them properly. This is a result of the SUE setting condition codes differently than the PDP-11. Byte instructions are the worst because the SUE does all arithmetic and logic using a full sixteen bits and sets the condition codes accordingly.

Execution also presents a problem with storing of the result. It is a problem because it must be determined not only whether the result is to be stored into a memory location or a register, but whether it is to be stored at all. The storing is performed by two routines, one for byte and one for word instructions. Those instructions such as CMP which do not restore the result bypass these routines.

To prevent the need for re-evaluating the destination mode the data address is always saved in a register, and to make it easier to determine if the mode was mode 0 all mode routines other than mode 0 routines set bit three in the instruction register. This destroys the actual mode, but it is no longer needed once it has been decoded. Thus after the execution

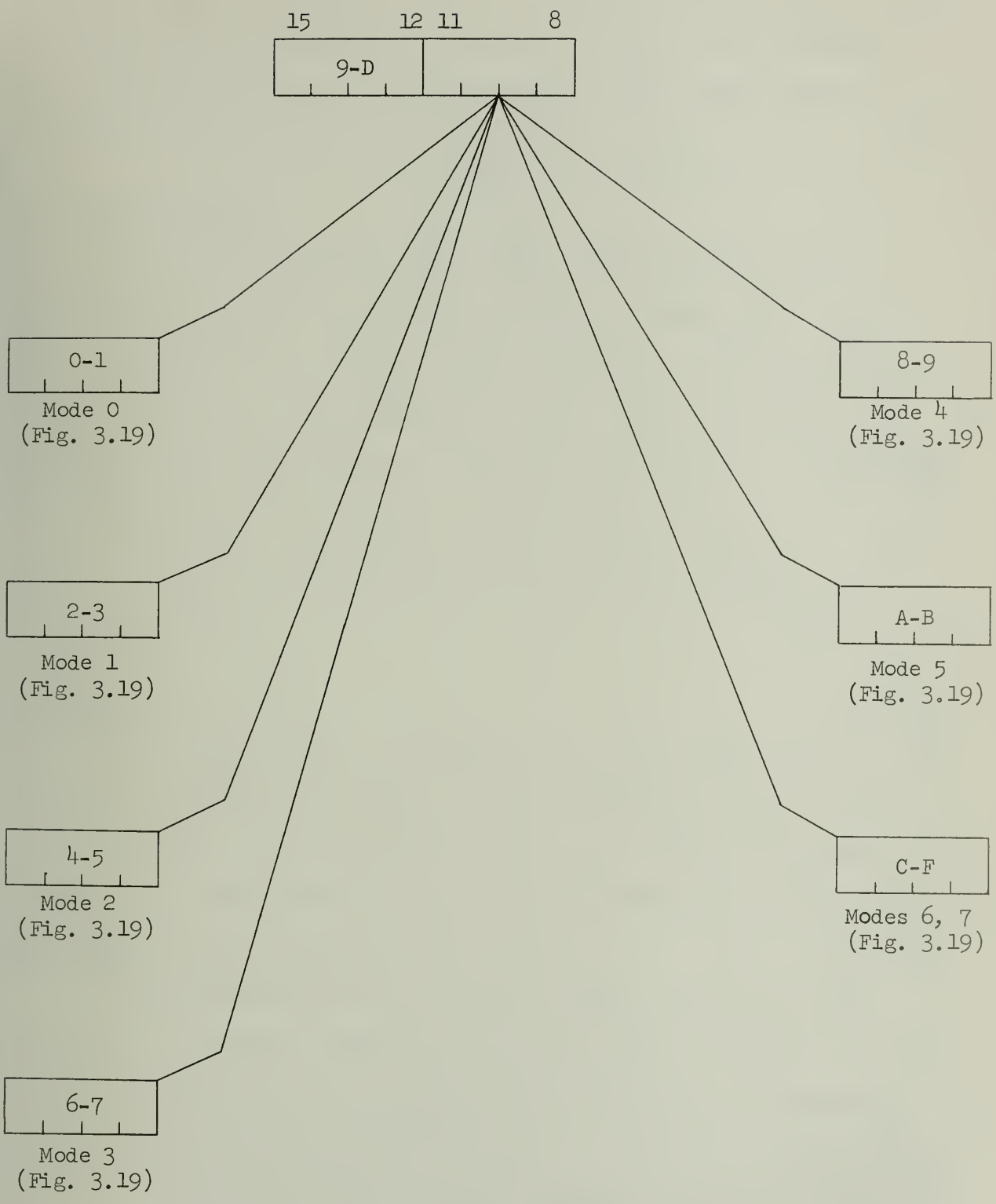


Figure 3.18 Double Operand Byte Instruction
Source Mode Decoding

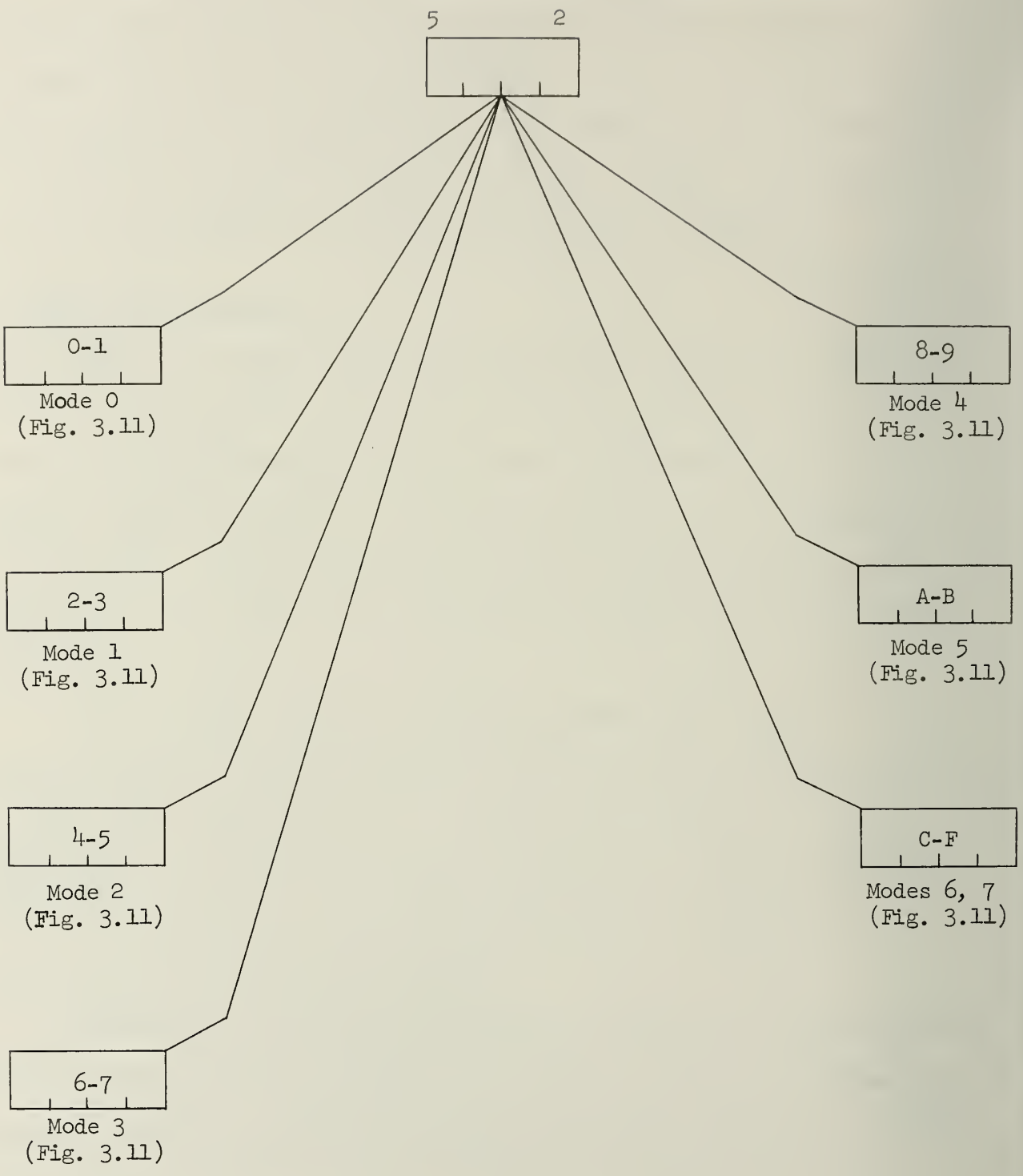


Figure 3.19 Double Operand Byte Instruction
Destination Mode Decoding

determining if the destination mode was 0 is simply a one bit test. If the destination mode is 0 a check must be made to determine if the register specified was the program counter. If it was special action which is described in the next section must be taken.

G. Instruction Fetch

To increase the speed of the emulator the next instruction is fetched while the current instruction is being executed whenever feasible. This is not done on any instruction which will result or possibly result in the program counter being changed such as in branch instructions and software interrupt instructions. Exceptions to this rule are instructions with a destination mode of 0 which specify register 7, the program counter. In these cases the next instruction is reread in case a change was made to the program counter.

H. Interrupts

There are two types of interrupts which occur on the PDP-11. These are software generated interrupts and I/O interrupts. An interrupt causes the old processor status and program counter to be stored on the stack and new ones to be loaded from a two word vector in memory. The address of the vector depends on the interrupt which occurs.

Software interrupts occur whenever an EMT, TRAP, IOT, or BPT instruction is executed. They also occur after each instruction is executed whenever the trace bit in the processor status word is set, or when an attempt is made to access an illegal address or execute an illegal instruction. The vector address for each software interrupt is predefined in the firmware to those specified for the PDP-11.

I/O interrupts occur whenever the bus controller determines that an I/O device is requesting an interrupt on a level which is not masked off by the emulator. When the interrupt bit is detected by the emulator it responds by enabling the interrupt resulting in the I/O device placing its device address on the bus so the CPU can read it. Unlike the PDP-11 the device address is given and not a vector address. Since the emulator requires a vector address it must be calculated from the device address. This is done by extracting bits 4, 5, 6, 7, 9, 10 and 11 and shifting them to result in the sequence

0 0 0 0 0 0 0 B11 B10 B9 B7 B6 B5 B4 0 0

This does not allow the vector address to be independent of the I/O device address as it is on a normal PDP-11, but this is the only way it can be done without making a hardware change.

Each time an interrupt occurs the old PS must be converted into the PDP-11 format and the new PS into the internal format. This conversion must also be done when returning from interrupts or accessing the processor status word directly. To enable the emulator to use the same set of conversion routines for all three cases the routines were implemented as subroutines. Since the SUE does not support subroutines, returns are performed using the table. An offset into the table is loaded into a register before the routine is called and is used to select the return address from the table. This is slow but necessary to avoid the large amount of control store which would be required if multiple routines were used.

CHAPTER IV

HARDWARE MODIFICATIONS TO IMPROVE PERFORMANCE

Although the SUE microprogram format is quite flexible and not highly dependent on the SUE's instruction set, the fact remains that it was not designed to allow easy emulation of the PDP-11 instruction set. As a result there are many minor changes which could be made to the hardware which would result in considerably improved performance with a smaller amount of microstore being required. Most of these changes can be logically implemented very easily while physically it is very difficult because the SUE processor uses a multilayered board. Some of these changes are discussed below.

A. Program Status Word

As described in Chapter III the internal format of the processor status word used by the emulator differs considerably from the format used by the PDP-11. This difference is the result of the hardware being designed for the status word as used by the SUE instruction set. The SUE uses a four bit mask for the processor priority. This mask occupies the top four bits of the status word. The PDP-11 uses a three bit encoded priority. It would be quite easy to implement a decoding circuit which would set the mask properly.

Another problem with the program status results from the fact that the position and order of the N, Z, C and V bits differ between the

SUE and the PDP-11. The hardware also requires that the N and Z bits be put into the status word at a different time from the C and V bits. This could easily be changed to allow setting of all four bits at the same time and in the same position as the PDP-11.

B. Condition Codes

Another difference between the SUE and the PDP-11 is that the condition codes are set differently in some operations. This is especially true with shift operations. The PDP-11 sets the overflow on all types of shifts while the SUE only sets the overflow on the arithmetic shift left. Thus to set the condition codes properly several tests may have to be performed.

C. Byte Instructions

Closely related to the condition code problem is that of byte instructions. ALU operations on the SUE are always a full sixteen bits. To set the condition codes properly on byte instructions requires considerable testing.

Another problem with byte instructions is that the SUE accesses bytes in the opposite order of the PDP-11. The PDP-11 accesses the low byte of a word as an even address while the SUE accesses the low byte as an odd address. This requires the emulator to complement the low address bit every time a byte is fetched from memory.

D. Subroutine Capabilities

Many routines in the emulator are repeated two or more times. This is especially true with address mode routines. If the SUE had a microprogram subroutine capability it would be possible to use the same set of routines for both single operand and double operand instructions. This would save a considerable amount of control store.

E. Exceptional Conditions

There are three conditions which must be tested for each instruction executed. These are: (1) test for pending interrupts, (2) test if the halt flip flop has been set, and (3) test if the trace bit is set. Each of these must be tested separately. It would be much faster if a test for any of these conditions could be performed with one microinstruction, and if one of them is set to test further to determine which one is set.

F. Registers

One of the biggest problems with the SUE is the lack of available registers. Four more registers should be added to the register file to increase the number from twelve to sixteen. It would also be desirable to be able to load the R register with the ALU output. This would allow register data and data from memory to be handled in the same way.

G. Octal Oriented Fields

Probably the hardest modification to make would be to make the fields oriented toward octal numbers rather than hexadecimal numbers. This would allow easier decoding with less table space.

LIST OF REFERENCES

[1] SUE Computer Handbook, Lockheed Electronics Company, Los Angeles, California, 1972.

[2] Microprogramming Guidelines for the SUE 1110 Micro-Processor, SUE Application Memo Number 101, Lockheed Electronics Company, Los Angeles, California, 1972.

[3] Snyder, F. G., The NPL 110 Processor, Interdepartmental Communication AS-31, Lockheed Electronics Company, Los Angeles, California, October 22, 1971.

[4] SUE 1110 Processor Logic Drawings, Lockheed Electronics Company, Los Angeles, California, 1972.

[5] PDP-11/20 Processor Handbook, Digital Equipment Corporation, Maynard, Massachusetts, 1971.

[6] PDP-11/40 Processor Handbook, Digital Equipment Corporation, Maynard, Massachusetts, 1972.

[7] PDP-11/45 Processor Handbook, Digital Equipment Corporation, Maynard, Massachusetts, 1971.

APPENDIX A

INSTRUCTION TIMING

Branch Instruction Timing

Mnemonic	No Branch	Forward	Backward
BR		1.77 us	1.80 us
BEQ	1.65 us	2.03 us	2.06 us
BNE	1.65 us	2.03 us	2.06 us
BMI	1.65 us	2.03 us	2.06 us
BPL	1.65 us	2.03 us	2.06 us
BCS	1.65 us	2.03 us	2.06 us
BCC	1.65 us	2.03 us	2.06 us
BVS	1.65 us	2.03 us	2.06 us
BVC	1.65 us	2.03 us	2.06 us
BLT	1.90 us	2.42 us	2.45 us
BGE	2.00 us	2.29 us	2.32 us
BLE	2.16 us	2.68 us	2.71 us
BGT	2.13 us	2.55 us	2.58 us

Double Operand Instruction Timing

Instruction Time = Execution Time + Source Time + Destination Time

Double Operand Execution Times

Mnemonic	Time	Mnemonic	Time
BIT	1.91 us	BITB	2.78 us
BIC	2.43 us	BICB	3.30 us
BIS	2.43 us	BISB	3.30 us
MOV	2.30 us	MOVB	2.69 us
CMP	2.30 us	CMPB	2.95 us
ADD	2.72 us	SUB	2.72 us
XOR	2.43 us (source mode is always 0)		

Double Operand Source and Destination Times

	Source		Destination	
Mode	Word	Byte	Word	Byte
0	0.39 us	0.39 us	0.39 us	0.52 us
1	0.91 us	0.91 us	1.13 us	1.00 us
2	0.91 us	1.04 us	1.13 us	1.56 us
3	1.59 us	1.46 us	1.98 us	1.85 us
4	0.78 us	1.30 us	1.13 us	1.56 us
5	1.59 us	1.46 us	1.98 us	1.85 us
6	1.59 us	1.46 us	1.98 us	1.85 us
7	2.47 us	2.31 us	2.83 us	2.70 us

Single Operand Instruction Timing

Instruction Time = Execution Time + Destination Time

Mnemonic	Time	Mnemonic	Time
CLR	2.82 us	CLRB	2.97 us
COM	3.11 us	COMB	3.21 us
TST	2.17 us	TSTB	2.17 us
INC	3.11 us	INCB	3.37 us
DEC	3.11 us	DECB	3.50 us
NEG	3.11 us	NEGB	3.89 us
ADC	3.27 us	ADCB	3.89 us
SBC	3.50 us	SBCB	4.02 us
ROL	3.60 us	ROLB	3.63 us
ROR	3.79 us	RORB	3.92 us
ASL	3.21 us	ASLB	3.92 us
ASR	3.66 us	ASRB	4.05 us
SWAB	3.99 us		

Single Operand Destination Modes

Mode	Word	Byte
0	0.39 us	0.52 us
1	1.00 us	1.00 us
2	1.20 us	1.46 us
3	1.81 us	1.85 us
4	1.46 us	1.59 us
5	2.04 us	1.85 us
6	1.81 us	1.95 us
7	2.70 us	2.70 us

JSR, JMP, and SXT Instruction Times

Instruction Time = Execution Time + Source Time

Execution Times

Mnemonic	Time
JSR	2.85 us
JMP	2.69 us
SXT	3.86 us

Destination Mode Times

Mode	Time
0	0.39 us
1	0.39 us
2	0.65 us
3	1.26 us
4	0.65 us
5	1.26 us
6	1.13 us
7	1.87 us

Miscellaneous Instructions

Mnemonic	Time	Mnemonic	Time
MARK	3.01 us	TRAP	7.77 us
SOB	2.00 us	EMT	7.77 us
RTS	2.85 us	BPT	8.68 us
RTI	4.50 us	IOT	8.68 us
RTT	4.50 us	SPL	2.69 us
STM	3.01 us for 1st register + 1.04 us for each additional register		
COM	3.14 us for 1st register + .85 us for each additional register		
BTM	2.01 us + 1.70 us for each word transferred		
BTS	2.01 us + 1.70 us for each word transferred		

Variable Time Instructions - HALT, WAIT

APPENDIX B

DETAILED SUE MICROCODE DESCRIPTION

35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S			T	A				C		D		X				F		Y		M				L2				L1				Z	W		

S field, sequential control

S = 0 sequential step

= 1 sequential step with special command designated by the L2 field

L2 = 0 Unused

= 1 Clear run flip flop (RBFS)

= 2 Load extended address bits 16 and 17 from low order bits of ALU output.

= 3 Permit interrupt - signals bus controller to have device place its device number on Infibus data lines.

= 4 AND C, V, A17, A16 to L1 field of next microinstruction. C is carry flip flop, V is overflow flip flop and A17, A16 are extended address bits.

= 5 AND N, Z, 0 to bits 6, 5, 4 in the L1 field. N is bit 15 of the T register, Z is the output of the ONES flip flop, and 0 is bit 0 of the T register.

L2 = 6 Set halt flip flop (HBFS)

= 7 Shift T register one bit. The type of shift is specified by the M field.

M = 0 Left arithmetic
= 1 Left linked logical
= 2 Left open logical
= 3 Left closed logical
= 4 Right arithmetic
= 5 Right linked logical
= 6 Right open logical
= 7 Right closed logical

I2 = 8 Reset bus access error flip flop (BAES)

= 9 Start full word read

= A Start full word write

= B Unused

= C Unused

= D Start byte read if bit 11 of the E register is set, otherwise start full word read.

= E Start byte write if bit 11 of the E register is set, otherwise start full word write.

= F Unused

S = 2 Unconditional branch. Branch occurs after next microinstruction. Branch address covers M, I2 and L1 fields.

S = 3 Bus access wait

If the run flip flop is set the processor waits until the bus access is completed before continuing to the next microinstruction. If a timeout occurs the next microinstruction is skipped and microprogram control is transferred to the eight bit address specified in the I2, L1 fields of the microinstruction.

Slave address recognition

If the run flip flop is not set the current microinstruction is executed repeatedly, once for each slave address recognition. Bus address bits one through four should be ANDed to the X field. This is specified by the F field (F = 3) in both the bus access wait microinstruction and the microinstruction immediately preceeding it. The A field should specify that the ALU output be equal to the XBUS input (A = F) and that it be written into both the file register and the T register (W = 6). Depending on whether the slave address recognition is for a read or a write, the A field is modified by the hardware so that for a read T = X is performed while for a write X = R is performed. This sequence is performed for each slave address recognition until the run flip flop is set manually via the control panel or by another processor.

S = 4,5 Jump if condition false, Jump if condition true

If the jump condition is met a jump is performed after the next microinstruction. The jump address is an eight bit address within the current page and is specified in the I2, L1 fields.

The jump condition is determined by the M field as follows.

M = 0 Halt flip flop set (HBFS)
1 No pending interrupts (NEXT)
2 Bit 0 of R register set (R00S)
3 NOR of bits 12 and 13 of E register
4 Ones flip flop set (ONES)
5 Carry flip flop set (CRYS)
6 Overflow flip flop set (OVFS)
7 Bus access error flip flop set (BAES)
8 Loop counter all ones (EMAX)
9 Bit 3 of E register set (E03S)
A Bit 7 of E register set (E 7S)
B Bit 11 of E register set (E11S)
C Bit 14 of E register set (E14S)
D Not used
E Bit 7 of E register set (E07S)
F Not used

S = 6 Jump if T bit is

A jump occurs after the next microinstruction if the bit in the T register specified by the M field is a 0. The eight bit jump address is given in the L2, L1 fields.

S = 7 Jump if T bit is 1

A jump occurs after the next microinstruction if the bit in the T register specified by the M field is a 1. The eight bit jump address is given in the L2, L1 fields.

D field, do next microinstruction control

Because the SUE fetches the next microinstruction while the current microinstruction is being executed branches and jumps do not occur on the next microinstruction but on the microinstruction following the next. It may be desirable to skip the next microinstruction depending on whether or not a conditional jump was taken. It may also be desirable to perform a test and skip or do the next microinstruction depending on the results of the test. The D field serves this purpose, allowing the user to conditionally skip or do the next microinstruction. The actual conditions of the skip depend on the contents of the S field.

If S = 0 through 3 and
D = 0 Do next microinstruction unconditionally
= 1 Skip next microinstruction if the output of the ALU is not all ones
= 2 Skip next microinstruction if the output of the ALU is all ones
= 3 Skip next microinstruction unconditionally

If S = 4 through 7 and
D = 0 Do next microinstruction unconditionally
= 1 Do next microinstruction if the jump is taken
= 2 Skip next microinstruction if the jump is taken
= 3 Skip next microinstruction unconditionally

T field, ALU operation type

T = 0 ALU operation is logical
= 1 ALU operation is arithmetic

C field, carry control

The C field controls the setting of the carry and overflow flip flops and also the carry input to the ALU on arithmetic operations. The exact meaning of the C field depends on the contents of the T and the A field.

If T = 0 and
C = 0 No change
= 1 Set carry flip flop
= 2 Clear carry and overflow flip flops
= 3 Not allowed

If T = 1 and
C = 0 Carry input to the ALU is zero, and carry and overflow flip flops are not changed.
C = 1 Carry flip flop is enabled to ALU carry input, and the results do not change the carry and overflow flip flops.
C = 2 Carry input to the ALU is zero, and the carry and overflow flip flops are set according to the result.
C = 3 Carry flip flop is enabled to ALU carry input and the carry and overflow flip flops are set according to the result.

If T = 1 and A is 2 or 6
C = 0 Carry input to the ALU is one, and the carry and overflow flip flops are not changed.
C = 1 Complement of carry flip flop is enabled to the ALU carry input, and the carry and overflow flip flops are not changed.
C = 2 Carry input to the ALU is one, and the ALU carry and overflow flip flops are set to the complement of what they would normally be set according to the result.
C = 3 Complement of the carry flip flop is enabled to the ALU carry input, and the carry and overflow are set to the complement of what they would normally be set according to the result.

A field, ALU operation

If the T field is zero the A field specifies one of the logical functions given in Table B.1 while if the T field is one the A field specifies one of the arithmetic functions listed in Table B.1.

A Field Code	Logical (T = 0)	Arithmetic (T = 1)
0	$\overline{X}$	X + C
1	$\overline{X \vee Y}$	(X ∨ Y) + C
2	$\overline{X} \cdot Y$	(X ∨ $\overline{Y}$) + C
3	Zero	ONES + C
4	$\overline{X \wedge Y}$	X + (X ∧ $\overline{Y}$) + C
5	$\overline{Y}$	(X ∨ Y) + (X ∧ $\overline{Y}$) + C
6	X ⊕ Y	X + $\overline{Y}$ + C
7	X ∧ $\overline{Y}$	(X ∧ $\overline{Y}$) + ONES + C
8	$\overline{X}$ ∨ Y	X + (X ∧ Y) + C
9	$\overline{X \oplus Y}$	X + Y + C
A	Y	(X ∨ $\overline{Y}$) + (X ∧ Y) + C
B	X ∧ Y	(X ∧ Y) + ONES + C
C	ONES	X + X + C
D	X ∨ $\overline{Y}$	(X ∨ Y) + X + C
E	X ∨ Y	(X ∨ $\overline{Y}$) + X + C
F	X	ONES + X + C

Table B.1 ALU Operations

X field, register file select

The X field selects one of twelve registers (0 through B) to be used as the X input to the ALU. If C through F are specified the X input will be zero. This field may be modified according to the F field of the previous microinstruction.

F field, next order X field modification

The F field is used to modify the X field of the following micro-instruction.

F = 0 Next X field unmodified
1 AND bits 0 through 3 of the E register to the next X field
2 AND bits 4 through 7 of the E register to the next X field
3 AND Infibus address bits 1 through 4 to the next X field

Y field - Y input select

The Y field is used to control the multiplexor which selects the Y input to the ALU.

Y = 0 R register
1 A register
2 T register
3 An eight bit literal in the L2, L1 fields is mapped into 16 bits according to the M field and used as the Y input. The mapping is given in Table B.2.

Bits	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
Literal / M code	L2	L1	L2	L1
0	X	X	X	X
1	X	X	X	
2	X	X		X
3	X	X		
4	X		X	X
5	X		X	
6	X			X
7	X			
8		X	X	X
9	X	X		
A		X		X
B		X		
C			X	X
D			X	
E				X
F				

Table B.2 Mapping of the literal field to the ALU Y input according to M field. X indicates the literal field is applied to the bits specified above.

W field - write ALU output select

The W field specifies which registers the output of the ALU is to be written into.

W = 0 Set the ones flip flop if the output of the ALU is all ones, clear it otherwise.
1 A register
2 T register
3 Loop counter (E register bits 0 through 3).
4 File register specified by X field as well as set ones flip flop.
5 File register specified by X field and A register.
6 File register specified by X field and T register.
7 File register specified by X field and E register.

Z field, special literal enable

If the Z field is one the L1 field is decoded to determine the type of special literal.

L1 bits 5, 4

= 0 Select table value. The L2 field and for extended tables the M field specify the most significant bits of the table address. Bits 7 and 6 specify a four bit field of the E register to be used as the least significant bits of the table address.

L1 bits 6,5

= 0 Use E register bits 0 through 3 for the least significant table address bits.
= 1 Use E register bits 4 through 7 for the least significant table address bits.
= 2 Use E register bits 8 through 11 for the least significant table address bits.
= 3 Use E register bits 12 through 15 for the least significant table address bits.

L1 bits 5, 4

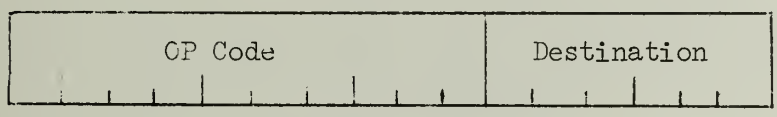
= 1 AND interrupt level of highest pending interrupt which is not masked off to L field bits 8, 7.
= 2 AND CPU number to L2 field bits 10, 9.
= 3 AND a four bit field of the E register as specified by L1 field bits 6, 5 to the M field of the next microinstruction. The selection of this field is the same as for L1 bits 5, 4 = 0.

APPENDIX C

DETAILED INSTRUCTION SET DESCRIPTION

This appendix gives a description of all instructions which have been implemented in the emulator. All numbers given are base 8 except where noted otherwise.

Single Operand Format Instruction



Single Operand Format

Mnemonic: CLR, CLRB
Opcode: 0050, 1050
Operation: Clears the contents of the destination.
Condition Codes:
N: Cleared
Z: Set
V: Cleared
C: Cleared

Mnemonic: COM, COMB
Opcode: 0051, 1051
Operation: Replaces the contents of the destination with their logical complement.
Condition Codes:
N: Set if the result is negative; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
V: Cleared
C: Set

Mnemonic: INC, INCB
Opcode: 0052, 1052
Operation: Adds one to the contents of the destination.
Condition Codes:
N: Set if the result is negative, cleared otherwise.
Z: Set if the result is zero, cleared otherwise.

V: Set if the destination was positive and the result is negative.
C: Not affected.

Mnemonic: DEC, DECB
Opcode: 0053, 1053
Operation: Subtracts one from the contents of the destination.
Condition Codes:
N: Set if the result is negative, cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
V: Set if the destination was negative and the result is positive, cleared otherwise.
C: Not affected.

Mnemonic: NEG, NEGB
Opcode: 0054, 1054
Operation: Replaces the contents of the destination by its two's complement.
Condition Codes:
N: Set if the result is negative; cleared otherwise.
Z: Set if the result is 0; cleared otherwise.
V: Set if the result is 100000_8; cleared otherwise.
C: Cleared if the result is 0; set otherwise.

Mnemonic: ADC, ADCB
Opcode: 0055, 1055
Operation: Adds the contents of the carry bit to the contents of the destination.
Condition Codes:
N: Set if the result is negative; cleared otherwise.
Z: Set if the result is 0; cleared otherwise.
V: Set if the contents of the destination were 077777_8 and the carry bit was set; cleared otherwise.
C: Set if the contents of the destination were 177777_8 and the carry bit was set; cleared otherwise.

Mnemonic SBC, SBCB
Opcode: 0056, 1056
Operation: Subtracts the contents of the carry bit from the contents of the destination.
Condition Codes:
N: Set if the result was negative, cleared otherwise.
Z: Set if the result was zero, cleared otherwise.
V: Set if the result is 100000_8; cleared otherwise.
C: Cleared if the result is 0 and the carry bit was set; cleared otherwise.

Mnemonic: TST, TSTB
Opcode: 0057, 1057
Operation: Sets the N and Z condition codes according to the contents of the destination.
Condition Codes:
- N: Set if the contents of the destination were negative; cleared otherwise.
- Z: Set if the contents of the destination were 0; cleared otherwise.
- V: Cleared.
- C: Cleared.

Mnemonic: ROR, RORB
Opcode: 0060, 1060
Operation: Rotates all bits of the destination contents one bit to the right. Bit 0 is loaded into the carry bit and the contents of the carry bit are loaded into the most significant bit of the destination.
Condition Codes:
- N: Set if the result is negative; cleared otherwise.
- Z: Set if the result is 0; cleared otherwise.
- V: Loaded with the Exclusive OR of the N and the C bit as set by the rotate operation.
- C: Loaded with the least significant bit of the destination.

Mnemonic: ROL, ROLB
Opcode: 0061, 1061
Operation: Rotates all bits of the destination one place to the left. Bit zero is loaded with the contents of the carry bit and the carry bit is loaded with the most significant bit of the destination.
Condition Codes:
- N: Set if the result is negative; cleared otherwise.
- Z: Set if the result is 0; cleared otherwise.
- V: Loaded with the Exclusive OR of the N and the C bit as set by the rotate operation.
- C: Loaded with the least significant bit of the destination.

Mnemonic: ASR, ASRB
Opcode: 0062, 1062
Operation: Shifts the destination operand right one place with the sign bit being replicated. The carry bit is loaded with the least significant bit of the destination.
Condition Codes:
- N: Set if the result is negative; cleared otherwise.
- Z: Set if the result is 0; cleared otherwise.
- V: Load with the Exclusive OR of the N and Z bits as set by the completion of the shift operation.
- C: Loaded with the most significant bit of the destination.

Mnemonic: SXT
Opcode: 0067
Operation: Sets all bits in the destination equal to the N bit in the PS.
Condition Codes: N: Unaffected.
Z: Set if N bit is cleared.
V: Unaffected.
C: Unaffected.

Mnemonic: SWAB
Opcode: 0003
Operation: Exchanges the high-order byte and low-order byte of the destination.
Condition Codes: N: Set if the low-order byte of the result is negative; cleared otherwise.
Z: Set if the low-order byte of the result is zero; cleared otherwise.
V: Cleared.
C: Cleared.

Mnemonic: MARK
Opcode: 0064
Operation: The destination field contains a six bit unsigned number. This value is multiplied by two and added to the stack pointer (R6). The program counter is then loaded with the contents of R5 and R5 is loaded with the contents of the location whose address is in R6. Two is then added to R6.
Condition Codes: Unaffected.

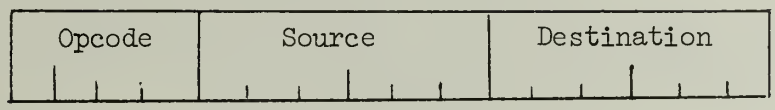
Mnemonic: JMP
Opcode: 0001
Operation: Places the destination address into the program counter. Register mode is illegal for this instruction.

Mnemonic: MTPI
Opcode: 0066
Operation: Calculates the destination address under the current user mode, then moves the top element of the stack into the location in the previous users instruction space addressed by the destination.
Condition Codes: N: Set if the result was negative; cleared otherwise.
Z: Set if the result was 0; cleared otherwise.
V: Cleared.
C: Unaffected.

Mnemonic: MFPI
Opcode: 0065
Operation: Calculates the source address in the current users address space then moves the contents of this address in the previous users address space onto the current users stack.

Condition Codes:
- N: Set if the result is negative; cleared otherwise.
- Z: Set if the result is 0; cleared otherwise.
- V: Cleared.
- C: Unaffected.

Double Operand Format Instructions



Mnemonic: MOV, MOVB
Opcode: 01, 11
Operation: Moves the source operand into the destination operand. On MOVB instructions which have a register for the destination the sign of the low byte is extended into the high byte.
Condition Codes:
- N: Set if result is negative; cleared otherwise.
- Z: Set if the result is 0; cleared otherwise.
- V: Cleared.
- C: Unaffected.

Mnemonic: CMP, CMPB
Opcode: 02, 12
Operation: Subtracts the destination from the source and sets the condition code accordingly. Neither the source nor the destination is changed.
Condition Codes:
- N: Set if the result is negative, cleared otherwise.
- Z: Set if the result is 0; cleared otherwise.
- V: Set if there was an arithmetic overflow; cleared otherwise.
- C: Cleared if there was a carry; set otherwise.

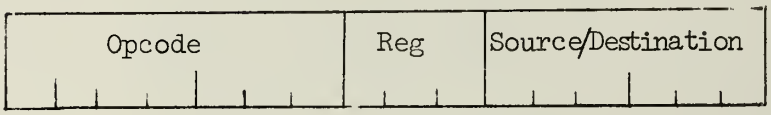
Mnemonic: BIT, BITB
Opcode: 03, 13
Operation: Sets the condition codes according to the result of the AND of the source and destination operands. Neither operand is changed.
Condition Codes:
- N: Set if the result is negative; cleared otherwise.
- Z: Set if the result is zero; cleared otherwise.
- V: Cleared.
- C: Unaffected.

Mnemonic: BIS, BISB
Opcode: 05, 15
Operation: Sets each bit in the destination that is set in the source.

Condition Codes:
- N: Set if the result is negative; cleared otherwise.
- Z: Set if the result is 0; cleared otherwise.
- V: Cleared.
- C: Unaffected.

Mnemonic: ADD
Opcode: 06
Operation: Subtracts the source operand from the destination operand and replaces the contents of the destination with the result.
Condition Codes:
- N: Set if the result was negative; cleared otherwise.
- Z: Set if the result was 0; cleared otherwise.
- V: Set if there was an arithmetic overflow; cleared otherwise.
- C: Cleared if there was a carry from the most significant bit; cleared otherwise.

Register Operand Format Instructions



Mnemonic: MUL
Opcode: 070
Operation: Multiplies the contents of the destination register by the source operand and places the result in the destination register R and register Rvl. If R is an odd register only the least significant sixteen bits are stored.
Condition Codes:
- N: Set if the result is negative; cleared otherwise.
- Z: Set if the result is 0; cleared otherwise.
- V: Cleared.
- C: Set if the result is less than -2^{15} or greater than or equal to $2^{15} - 1$.

Mnemonic: DIV
Opcode: 071
Operation: Divides the thirty-two bit number in R and Rvl by the source operand. The quotient is placed in R and the remainder in Rvl.
Condition Codes:
- N: Set if the quotient is negative; cleared otherwise.
- Z: Set if the quotient is 0; cleared otherwise.
- V: Set if the source operand is zero or if the quotient would exceed 15 bits.
- C: Set if the source was zero.

Mnemonic: ASH
Opcode: 072
Operation: Shifts the contents of the register n places to the left or n places to the right where n is a six bit signed integer stored in the source field. If the integer is negative the register is shifted right and if it is positive it is shifted left.
Condition Codes:
N: Set if the result was negative; cleared otherwise.
Z: Set if the result was 0; cleared otherwise.
V: Set if the sign of the register changed during the shift; cleared otherwise.
C: Loaded with the last bit to be shifted out of the register.

Mnemonic: ASHC
Opcode: 073
Operation: Performs an arithmetic shift of n places on the contents of the registers R and Rvl treated as a thirty-two bit operand. The shift count n is specified in the source operand field which has a range of -32 to +31. A negative shift is a right shift and a positive shift is left.
Condition Codes:
N: Set if the result is negative; cleared otherwise.
Z: Set if the result is 0; cleared otherwise.
V: Set if the sign bit changes during the shift; cleared otherwise.
C: Loaded with the last bit shifted out of the thirty-two bit operand.

Mnemonic: XOR
Opcode: 074
Operation: The contents of the destination operand are replaced by the exclusive OR of the register and the destination operand.
Condition Codes:
N: Set if the result is negative; cleared otherwise.
Z: Set if the result is 0; cleared otherwise.
V: Cleared.
C: Unaffected.

Mnemonic: JSR
Opcode: 004
Operation: Jump to Subroutine. This instruction determines the destination address then saves the register specified on the processor stack, places the old PC into the specified register and places the destination address in the PC. Mode 0 is illegal for this instruction.

Mnemonic: SOB
Opcode: 077
Operation: Subtracts one from the specified register and if the register is not 0 a six bit unsigned number stored in the least significant six bits of the instruction is multiplied by two and subtracted from the current program counter.

Branch Format Instructions

The Branch instructions of the PDP-11 instruction set provide a means of doing unconditional and conditional transfers of program control to within -128 to +127 words of the current program counter. The new address is calculated by adding twice the offset to the current program counter. Condition codes are unaffected by branch instructions.

Mnemonic: BR
Opcode: 0004
Operation: Unconditional branch.

Mnemonic: BNE
Opcode: 0010
Operation: Branch if the Z bit in the processor status word is 0.

Mnemonic: BEQ
Opcode: 0014
Operation: Branch if the Z bit in the processor status word is 1.

Mnemonic: BGE
Opcode: 0020
Operation: Branch if the N and V bits in the processor status word are the same.

Mnemonic: BLT
Opcode: 0024
Operation: Branch if the N and V bits in the processor status word are different.

Mnemonic: BGT
Opcode: 0030
Operation: Branch if the Z bit is 0 and the N and V bits in the processor status word are the same.

Mnemonic: BLE
Opcode: 0034
Operation: Branch if the Z bit is 1 or if the N and V bits are different.

Mnemonic: BPL
Opcode: 1000
Operation: Branch if the N bit in the processor status word is 0.

Mnemonic: BMI
Opcode: 1004
Operation: Branch if the N bit in the processor status word is 1.

Mnemonic: BVC
Opcode: 1020
Operation: Branch if the V bit in the processor status word is 0.

Mnemonic: BVS
Opcode: 1024
Operation: Branch if the V bit in the processor status word is set.

Mnemonic: BCC
Opcode: 1030
Operation: Branch if the C bit in the processor status word is 0.

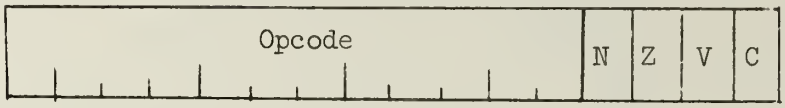
Mnemonic: BCS
Opcode: 1034
Operation: Branch if the C bit in the processor status word is set.

The EMT and TRAP instructions use the same format as the branch instructions with the exception that the offset field contains a function code.

Mnemonic: EMT
Opcode: 1040
Operation: Causes a software interrupt. The old processor status and program counter are saved on the processor stack, and the new program counter and processor status are loaded from locations 30 and 32.
Condition Codes: Loaded from location 32.

Mnemonic: TRAP
Opcode: 1044
Operation: Causes a software interrupt. The old processor status and program counter are saved on the processor stack, and the new program counter and processor status are loaded from locations 34 and 36.
Condition Codes: Loaded from location 36.

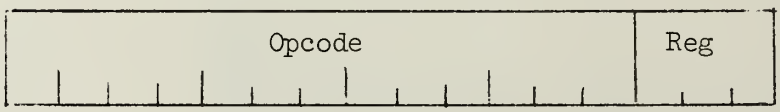
Condition Code Format Instructions



Mnemonic: CLC, CLV, CLZ, CLN, CCC, NOP
Opcode: 00024
Operation:
CLC - Clear C bit in the PS.
CLV - Clear V bit in the PS.
CLZ - Clear Z bit in the PS.
CLN - Clear N bit in the PS.
CCC - Clear all condition code bits in the PS.
NOP - Clear none of the condition codes in the PS (no operation).

Mnemonic: SEC, SEV, SEZ, SEN, SCC
Opcode: 00026
Operation:
SEC - Set C bit in the PS.
SEV - Set V bit in the PS.
SEZ - Set Z bit in the PS.
SEN - Set N bit in the PS.
SCC - Set all condition code bits in the PS.

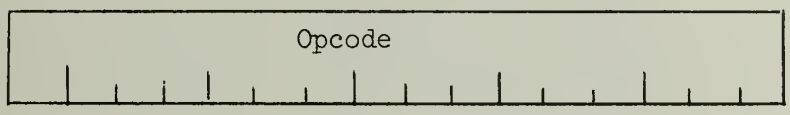
Subroutine Return Format Instructions



Mnemonic: RTS
Opcode: 00020
Operation: Loads the contents of the specified register into the program counter and then pops the top element of the stack into the register.

Mnemonic: SPL
Opcode: 00023
Operation: Loads the priority specified in the three bit register field into the priority bits of the processor status word. This instruction is only executed if the current mode is 00. In other cases it operates as a no-op.

Miscellaneous Instructions



Mnemonic: HALT
Opcode: 000000
Operation: Causes the processor to stop executing instructions. This instruction causes an illegal instruction trap if the current mode is not 00.

Mnemonic: WAIT
Opcode: 000001
Operation: Causes the processor to pause until an external interrupt occurs.

Mnemonic: RTI
Opcode: 000002
Operation: Pops the top of the processor stack and loads it into the program counter, and then it pops it again and loads the processor status word. This instruction is used to return from an interrupt procedure.

Mnemonic: BPT
Opcode: 000003
Operation: Causes a software interrupt through vector location 14.
Condition Codes: Loaded from location 16.

Mnemonic: IOT
Opcode: 000004
Operation: Causes a software interrupt through vector location 20.
Condition Codes: Loaded from location 22.

Mnemonic: RTT
Opcode: 000006
Operation: Same as RTI instruction except that no trace trap will occur after the execution of this instruction even if the T bit is set in the program status word.

Triple Operand Register Format Instructions

Mnemonic: STM
Opcode: 076
Operation: This instruction performs multiple register stores onto a stack. The number of the first register to be stored is specified in the leftmost register field, the number of the last register to be stored is specified in the second field, and the stack onto which the registers are to be stored is specified in the third field. When storing the stack register is autodecremented before storing each register. The number of the register being stored is treated as a MOD 8 number thus if the first register has a higher number than the last register R7 is stored followed by R0.

Mnemonic: LDM
Opcode: 075
Operation: The load multiple instruction is used to reload registers from a stack. The first register to be loaded is specified in the leftmost register field, the last register in the next field, and the stack register in the rightmost field. The registers are loaded in reverse order with the register number being treated as MOD 8. Thus if the number of the first register to be loaded is less than the number of the last register, R is loaded followed by R7. After each register is loaded the stack register is incremented by two.

Mnemonic: BTS
Opcode: 007
Operation: This instruction is used to transfer blocks of memory onto a stack. The leftmost register field specifies a register which contains the number of words to be transferred. The second register field specifies the source register, with the source mode always being the autoincrement mode. The destination register is specified in the third field, and the destination mode is always autodecrement mode.

Mnemonic:	BTM
Opcode:	107
Operation:	This instruction is the same as BTS with the exception that the destination mode is always autoincrement.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-74-621	2.	3. Recipient's Accession No.
4. Title and Subtitle Emulation of the PDP-11 Instruction Set on the Lockheed SUE Processor			5. Report Date April 1974 6.
7. Author(s) James Fredrick Hart			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. US NSF GJ36265
12. Sponsoring Organization Name and Address National Science Foundation Washington, D.C.			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

This thesis describes the emulation of the PDP-11 instruction set on the Lockheed SUE minicomputer. The PDP-11 instruction set is very powerful but the hardware is limited to single processor configurations. The SUE provides for multiprocessing as well as a very flexible microprogramming capability. The PDP-11 emulator allows these capabilities to be combined with the PDP-11 instruction set to provide an extremely powerful multiprocessing configuration

17. Key Words and Document Analysis. 17a. Descriptors

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page UNCLASSIFIED	22. Price